DEADLOCK-FREE SHARING OF RESOURCES
IN ASYNCHRONOUS SYSTEMS

Prakash G. Hebalkar

September 1970

PROJECT MAC

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Cambridge Massachusetts 02139

DEADLOCK-FREE SHARING OF RESOURCES IN ASYNCHRONOUS SYSTEMS*

Abstract

Whenever resources are shared among several activities that hoard resources, the activities can attain a state of deadlock in which, for lack of resources, none of the activities can proceed. Deadlocks can be prevented by coordination of the sharing. Efficient running of the activities under such coordination requires knowledge of the patterns of use of resources by the activities.

This thesis presents a study of deadlock prevention in systems in which a knowledge of the usage of resources by the activities during several phases of steady resource usage is available. A representation called a demand graph is presented and used for the study of deadlocks. The model is a general one and encompasses systems in which the activities themselves consist of more than one sequence of phases and are not necessarily independent of each other. The analysis is applicable to computer systems as well as systems in the realm of operations research.

*This report reproduces a thesis os the same title submitted to the Department of Electrical Engineering, Massachusetts Institute of Technology, in partial fulfillment of the requirements for the degree of Doctor of Science, September 1970.

ACKNOWLEDGMENT

The author would like to place on record his gratitude to his thesis supervisor, Professor J. B. Dennis, for excellent guidance. Prof. Dennis's comments have resulted in a substantial improvement in the quality of the presentation. He is also to be thanked for encouragement and support throughout the author's graduate studies.

The editorial comments of Professors F. J. Corbato and C. L. Liu, both readers, are greatly appreciated. Prof. Liu also deserves especial thanks for his cheerful optimism and friendly counsel during the many years of graduate study.

The Computation Structures Group at Project MAC has contributed much, in an intangible way, to the author's education. The author's office-mates Suhas Patil, Murray Edelberg and Lawrence Seligman provided valuable interaction. Suhas Patil is also to be thanked for reading the first draft of the thesis.

Miss Rubin is to be congratulated for doing an excellent job of typing the thesis under considerable pressure of time.

The author is grateful to Project MAC for financial support.

Work reported herein was supported in part by Project MAC, an M.I.T. research project sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Nonr-4102(01).

TABLE OF CONTENTS

1. Introduction 5

2. Demand Graphs for Systems with a Single Type of Resource 18

3. Demand Graphs for Systems with Several Types of Resources 50

4. Demand Graphs for Systems with Interacting and Internally Parallel Activities 100

5. Loops and Decisions 141

6. Conclusion 156

Appendix. Application of the Theory of Linear Inequalities to Demand Graphs 166

References 183

Index 185

Biographical Note 186

Introduction

Chapter 1

§1.1 Deadlocks

As this thesis deals with deadlocks and their prevention, it is necessary for the reader to appreciate the nature of deadlocks. Three examples are presented below, with the aim of introducing the concept of deadlock to the reader.

The first example concerns a canal with locks and two drawbridges on it. The drawbridges lie on a road, as shown in Figure 1.1, which has been laid so as to avoid a marsh and crosses the canal twice. Both the canal and the road carry traffic in one direction only. The principal traffic on the canal consists of barges. As a barge approaches Bridge A, a warning is sounded when the barge is 100 metres from the bridge and, when the bridge is free of cars, it is drawn. The bridge stays drawn until the tail end of the barge has passed the bridge. A similar discipline is followed for Bridge B.

The system works very well until a rather long barge comes in on a day when traffic is heavy. Then it can happen that Bridge A is drawn and a queue of cars begins to build up that extends well past Bridge B. Then the barge reaches Bridge B while its tail end is still under Bridge A. But Bridge B cannot be drawn because there are cars on it! The cars on Bridge B cannot move ahead until Bridge A is lowered and that cannot be done until the barge has moved ahead, which in turn cannot be done until the cars on Bridge B move on! A deadlock has thus occurred because neither the cars nor the barge can back up. The deadlock will persist indefinitely.

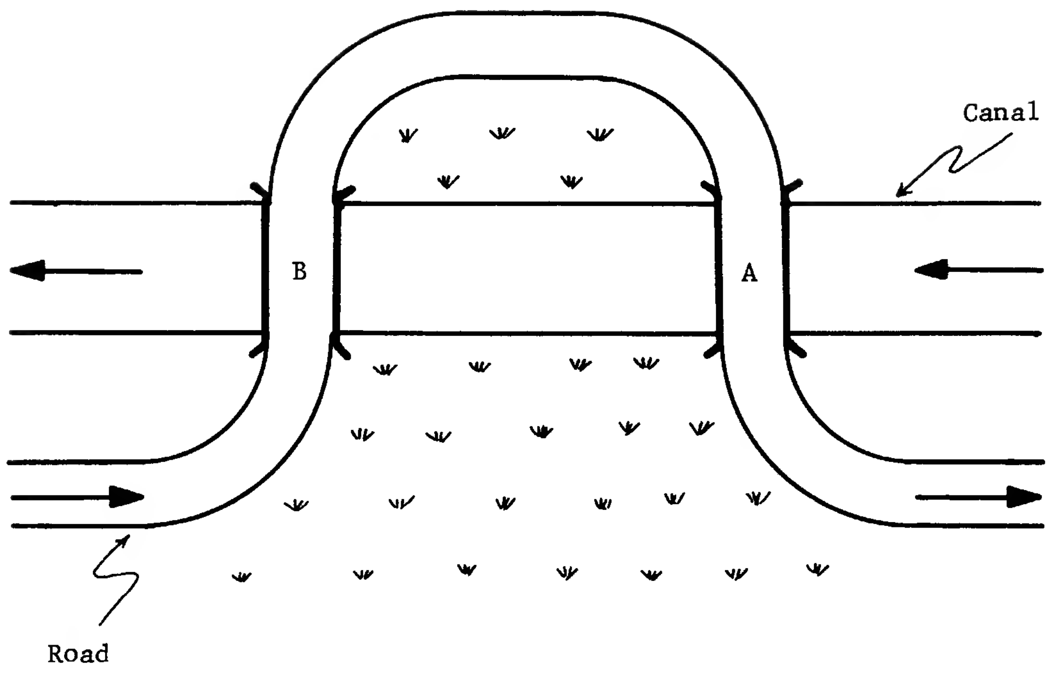


Figure 1.1

The deadlock above occurred because of improper planning of the use of the bridges by cars and barges. If the warning for Bridge B had been issued at the same time that it was issued for Bridge A, the deadlock would not have occurred. This is not just a matter of hindsight; rather, it indicates that deadlocks cannot be prevented without a priori knowledge of the use of shared resources (in this case the bridge). It will be noted that a stochastic model is useless in this case; knowing that the probabilities of there being very heavy traffic when a barge crosses the section of the canal between Bridges A and B, and that a barge is long enough to cause trouble, are each 0.07, with a consequent 0.995 probability (assuming independence of the two events) of successful operation, is of little comfort. Deadlocks, when they are catastrophic in their consequences, must be prevented.

The second example concerns a maintenance hangar for aeroplanes. The planes that come in for servicing represent tasks for the workshop. Planes coming in for servicing are put onto stands for service. It is not known, when a plane comes in, how much work needs to be done on it and, therefore, how long it will take to overhaul the plane. When a plane is taken in, the bottom of the plane is opened up on the stand and various kinds of jigs are inserted for the overhauling. If planes are taken in whenever a stand is empty, it is possible to reach a condition in which the jigs are all used up and yet each plane needs more jigs before its overhauling is complete and jigs are released. (It is assumed that jigs cannot be pulled off incompletely serviced planes as they also perform the structural function that parts that have been removed perform.) Once more deadlock is possible. The point being emphasized here is that the scheduling of work for the hangar is not analogous to that of scheduling work for an assembly shop. The servicing of planes is asynchronous, in the sense that the times for processing of planes are not the same. Thus the principal interest is not in picking a schedule that minimizes the average processing time but rather in letting the processing of jobs which are accepted proceed at their own pace, subject to the avoidance of deadlock. In this respect, the systems considered in this thesis differ fundamentally from the systems analyzed in the field of Project Scheduling as typified by [1]. Another fundamental difference is that resources (here, jigs) are not always returned between two overhauling operations, i.e., it is not true that at the end of an operation, all the resources required for its execution become available for general use. This retention of resources is a sine qua non for the occurrence of deadlocks and its absence in Job Shop Systems is probably why, to the best of the author's knowledge, it has not been studied in the field of Job Shop Scheduling. Job Shop Scheduling will be taken up later, in greater detail, at the end of Chapter 3.

The third example relates to computer systems with a one-level memory and multiprocessing. Here core memory is shared and processes can become deadlocked for lack of free core. Processes cannot be deprived of memory already allocated, as this implies nullification of any partial computation already performed. The penalty for de-allocation is thus the expenditure of time and computational effort to recompute, and this can be substantial. This example brings out the large cost that undoing the consequences of deadlock can imply, if at all this is possible. That deadlocks could be resolved in this example is not unusual. Deadlocks can almost always be resolved by preemption. Even in the first example the deadlock can be broken at the cost of the destruction of the cars on Bridge B. The resolution of deadlocks is no solution at all precisely because the price paid is too high to ignore the possibility of the prevention of deadlocks.

The problem of prevention of deadlocks has been approached recently with a view to seeking elegant solutions. Some of the earlier work is described below.

§1.2 Past Work

The best known past work in this field is that of Habermann [2,3] who extended the somewhat more specialized analysis that was given by Dijkstra in [4]. Habermann's analysis is summarized in the next paragraph and the one following it. However, both assume the availability of some information about the amounts of resource that will be needed by the different tasks in the system. Havender, in [5], treats a somewhat more specialized case of resource usage. The work of Habermann is the most elaborate of the three and also provides the basis and some of the terminology of this thesis.

Habermann considers sequential processes, i.e. tasks, (say m of them) sharing several (say n) types of resource. All the units of resource of any one type are equally useful. Each process is required to state the maximum amount of resource of each type that it will need — m_{ij} for process i and resource-type j. The processes are free to acquire and release resources as they please, subject to these maxima. The analysis assumes that the various maximum amounts, m_{ij}, for a process may be needed simultaneously, and thus there is a maximum demand vector for each process, $\underline{m}_i$ for process i, whose n components are the maximum demands for each of the resource types. Allocation of resources is done on the basis of actual requests for additional resource from processes and so as to prevent the occurrence of deadlock. At any instant, each process has been allotted a certain quantity of each kind of resource so that there is a vector of allocations to the process. Allocation vectors are represented by $\underline{a}_i$ (for the i^{th} process). The combined status of the processes at any time is thus represented by the allocation state, $\langle a_1, a_2, \ldots a_m \rangle$, whose components are the m vectors of allocation for the m processes. An allocation state is said to be safe if there is some sequence in which the needs of each process can be met, one at a time, so that all the processes can terminate. Each process is assumed to terminate within a finite amount of time once its needs have been satisfied fully. Habermann has shown that the definition of safeness can be restated as a test for the safeness of an allocation state, viz that there should exist a sequence, $i_1, i_2, \ldots i_m$ of the m processes so that the allocation vectors and the unused resources vector, $\underline{R}$, satisfy the set of inequalities:

$$\underline{m}_{i_1} - \underline{a}_{i_1} \le \underline{R}$$

$$\underline{m}_{i_2} - \underline{a}_{i_2} \le \underline{R} + \underline{a}_{i_1}$$

$$\cdots\cdots\cdots\cdots\cdots$$

$$\underline{m}_{i_m} - \underline{a}_{i_m} \le \underline{R} + \underline{a}_{i_1} + \underline{a}_{i_2} + \ldots \underline{a}_{i_{m-1}}$$

Habermann has shown that deadlock can be avoided if the allocation state is safe but not otherwise. It can be seen easily that the inequalities above can be rearranged in a canonical order so that the left hand sides are non-decreasing. Thus the unused resources, $\underline{R}$, at any time need only be as large as the smallest of the unsatisfied resource needs of the users at that time! (Clearly then the amount of unused resources need never exceed the smallest of the maximum demands, $\underline{m}_i$.)

In contrast to the good utilization of resources that is found above, when no information about resource usage is available at all, the processes can only be run sequentially. The greater information available in the former case is what permits more efficient utilization of resources. This is what suggests that systems capable of handling more detailed information about resource usage should be of interest, as even better utilization of resources may be possible. This thesis is an attempt to study how this more detailed information can be used to advantage.

Shoshani has worked on an extension of Habermann's analysis using an algebraic model [6]. His results are similar to, though somewhat less general than, some of the results obtained independently by the author and reported here. In [7] he discusses the problem of recovery from deadlocks with minimum cost and presents an elegant solution.

§1.3 The Problem

The systems dealt with in this thesis consist of a number of processes, i.e., unified sequences of activities. The processes are asynchronous, i.e., temporal relationships between the activities of two processes based on a single time axis are meaningless. The processes share several kinds of resource from a pool. The various combinations of resources needed during the activity of each process are assumed to be known[+]. The processes do not have to be sequential in activity or independent of each other[+]. The problem treated of is that of allocating resources in such a system in a manner that prevents the occurrence of deadlock and optimizes utilization of the resources. The choice of an appropriate model is important for the analysis of deadlock and a graphical one has been chosen for this purpose.

As before, an example is presented here which, it is hoped, will prove useful in gaining the proper perspective. The example will be referred to as "the construction analogue" later on, as it deals with the building construction industry and as the principal context for the treatment of deadlock prevention will be that of computer systems.

The construction analogue concerns a construction equipment rental company. Several contractors rent equipment from this company, the only

---

[+] These are the two areas in which Habermann's analysis is extended here.

one in the neighborhood, to build buildings which they sell when completed. From the point of view of the company, each contractor is a process which it serves. Each contractor knows the phases that his work will go through, such as foundation building, wall erection, and so on, and the amounts of each kind of equipment that he needs in each phase. He knows that when he needs bulldozers he does not need scaffolding, and so on, so that the maximum needs for each kind of resource (equipment) do not, in general, occur simultaneously. He does not know exactly how long each phase will last, because of uncertainties of weather, material supply and availability of labor. Moreover, these uncertainties are different for different contractors and so the different processes in the system are asynchronous. Each contractor gives the company a description of resource needs in phases and expects, in turn, to be rented equipment on a first-come-first-served basis but without ever being deadlocked in conjunction with other contractors. He will return equipment when he does not need it, but not under any other circumstances; for he works in competition with other contractors. Several contractors may undertake joint projects, so that their activities are not necessarily independent. Moreover, a single contractor may undertake several projects which can proceed independently of each other or interact at arbitrary points in their activity. Also, a phase in a contractor's activity, or a set of phases for that matter, may be capable of execution with more than one alternative combinations of equipment. Contractors are free to undertake new projects upon completion of others and new contractors can enter the system. The problem that the company faces is that of maximizing its income from the rental of the equipment, while satisfying all its clients.

In terms of computer systems, computations correspond to the contractors. New computations enter the system when they are created by the principals (users) of the system. The computations need not be sequential. The resources shared are active memory, arithmetic units, input output devices, etc. There is considerable latitude in the detail to which the analysis may be extended -- thus specialized functional units inside the arithmetic unit, for instance, can also be considered resources if it is so desired. The active memory is considered to consist of one level and space in it is allocated to processes dynamically. As the memory has only one level, it is not possible to free space in active memory by preemption without destroying information. When the memory does consist of several levels, deadlock cannot occur on account of memory. For free space can be created by moving information to a lower level; however, the large time delays in such movement of information that are encountered in practice emphasize the need for prevention of deadlocks, as does the possibility of thrashing. The inability to preempt resources is more evident in the case of input output devices such as tape-drives, plotters and graphic output devices.

It is not proposed that a user or programmer supply the information about resource needs; rather, it is assumed that a pre-processor of some sort, perhaps a compiler, provides the information. It is not a fanciful idea to expect that such information can be extracted from programs. It is already known how to get upper bounds on core usage of non-recursive programs, if only conservative estimates. It is not necessary to extract further detailed information from a program although, if such information can be obtained, it can be used. It is merely required of the principal that he state which procedures are used, and in what sequence, in the definition of the computation. Thus, rather than determining the largest of the memory requirements of the individual procedures making up the computation and stating just that, the entire information consisting of the sequence of procedure calls and the memory requirements of each procedure can be made available.

An important restriction that is placed on programs to which the study undertaken in this thesis applies is that they not contain unrestricted recursion; for it is impossible to guarantee that deadlocks will be prevented if the demands of a process can increase beyond bound.

§1.4 Plan of the Thesis

Chapter 2 introduces the demand graph as the model to be used to represent the systems of interest. Specialized demand graphs of systems with sequential processes and a single type of resource are analyzed here. A non-enumerative algorithm is presented for determination of safeness, a concept related to deadlock avoidance, in this chapter.

Chapter 3 extends the analysis of Chapter 2 to systems with more than one type of resource. The concepts of limited-backtracking and linearity are introduced and it is shown that linear algorithms for determination of safeness do not exist. The algorithm of Chapter 2 is also extended.

Chapter 4 introduces interactions between processes into the picture. The analysis of Chapter 3 is extended to this case.

An initial attempt at the handling of decisions, loops and alternative ways of satisfying the resource requirements of a process is made in Chapter 5.

Chapter 6 presents some concluding thoughts, and the appendix describes some properties of demand graphs deduced by the use of the theory of linear inequalities.

[illegible] Graphs For Systems With

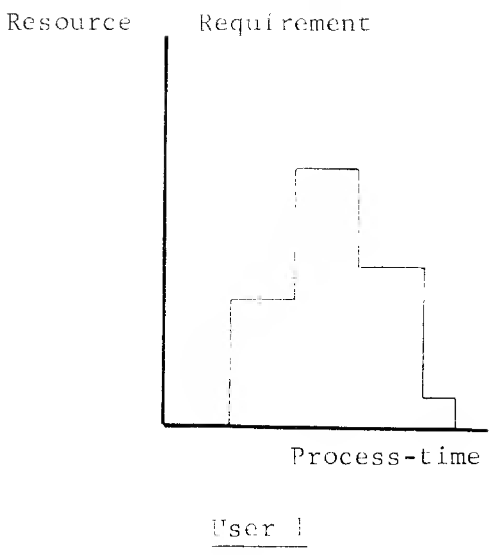
a

Single Type Of Resource

Chapter

§2.1 Problems of the Use of Continuous Time

It was pointed out in Chapter 1 that the systems of processes being investigated in this thesis are those in which information about the usage of resources during the activity of each process is available. A natural way to think about such information is as graphs of resource usage with time. Figure 2.1 illustrates such graphs for two processes which share one kind of resource. Unfortunately, such graphs use time axes which are meaningful only for the respective processes; for the processes are asynchronous and so no temporal relationships between the activities of two processes that are based on a single time axis can be defined. The graphs are thus incomparable. However, from the point of view of resource allocation, only the epochs corresponding to changes in resource usage are of interest -- the length of time, on any axis, between such epochs is irrelevant. Thus, only these epochs need to be represented in an abstract model for the study of deadlocks and resource allocation. The next section describes such a representation, viz the demand graph. The concept of a demand graph was inspired by Holt's work [8] on the representation of events and by the realization that it is the class of events, which correspond to changes in resource usage, that is of interest in the study of deadlocks.


Resource Requirement
Process-time
User 1

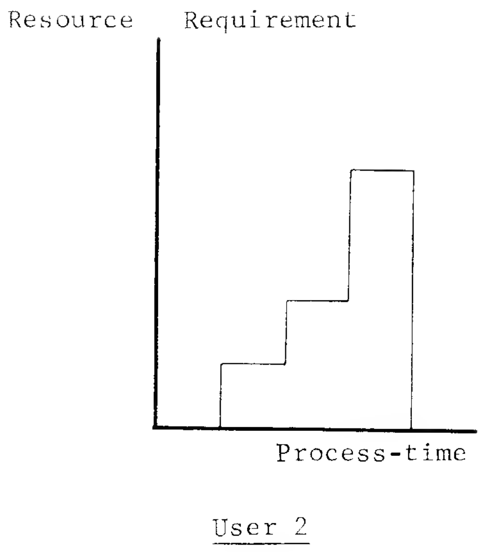

Resource Requirement
Process-time
User 2

Figure 2.1

§2.2 Demand Graphs

A demand graph is a finite directed graph with arcs and nodes; the nodes are called transitions. Associated with each arc is a quantity called a demand, chosen from a set Δ. A quantity called the capacity, which is represented by C and also chosen from the set Δ, is associated with the demand graph. The set Δ is ordered (partially or totally[†]) and the demands associated with the arcs of a demand graph are always less than or equal to the capacity associated with the demand graph. Demand graphs are generally dis-connected. In any case, every component of a demand graph must contain at least onc node that has in-degree zero and one node that has out-degree zero.

The study of demand graphs in this thesis will proceed from a restricted class of demand graphs, called Rectilinear Scalar Demand Graphs and studied in this chapter, to progressively less restricted classes.

§2.3 Rectilinear Scalar Demand Graphs

Rectilinear Scalar Demand Graphs, or Scalar Demand Graphs for brevity, are acyclic demand graphs that have the property that the components arc unilateral, i.e. for every pair of transitions at least one transition is reachable from the other by a path. Thc components thus look like

[†]See §2.7.

chains and for this reason they are formally termed chains. The section of a chain between any two transitions will be called a segment of the chain; clearly, an arc of a chain is a segment of that chain. The demands associated with the arcs of simple demand graphs belong to the set of non-negative integers, and so does the capacity, C, associated with the system. The demands associated with the first and last arcs of each chain are 0. These arcs are called initial and terminal arcs of the chains, respectively.

The Scalar Demand Graph is a model for a class of systems of processes in which resources are shared. The chains of a Scalar Demand Graph correspond to processes in the system represented by the graph. The transitions correspond to the epochs at which a change in resource usage occurs and the arcs to phases of activity of the processes, i.e. periods of steady resource usage. The processes can be said to be sequential as each phase can be followed by exactly one other phase. Moreover, as the sub-graphs consisting of chains are disjoint from each other, the processes they model can be said to be independent. The only interaction between processes is that due to sharing of resources. Later chapters will contain discussions that relate to broader classes of systems in which the processes are not so constrained. The demands associated with the arcs of the graph represent the demands for resources associated with the corresponding phases (of activity) of the processes. As the demands are integers, the processes modelled share a single type of resource from a common pool. The capacity, C, associated with a demand graph represents the size of this pool or the number of servers in this pool. The different servers are identical in their capability to serve and thus the resource can be said to be homogenous — in fact, a resource of any one type will always be considered to be homogeneous. The requirement that adjacent arcs have distinct demands is consistent with the fact that the transitions represent changes in resource usage. Needless to say, the resources are shared in an unpreemptable manner so that deadlocks can occur. The zero demands associated with the initial and final arcs of each chain represent the fact that processes which are uninitiated or terminated require no resource.

Some of the notation to be used in the discussion which follows is described next.

§2.4 Notation

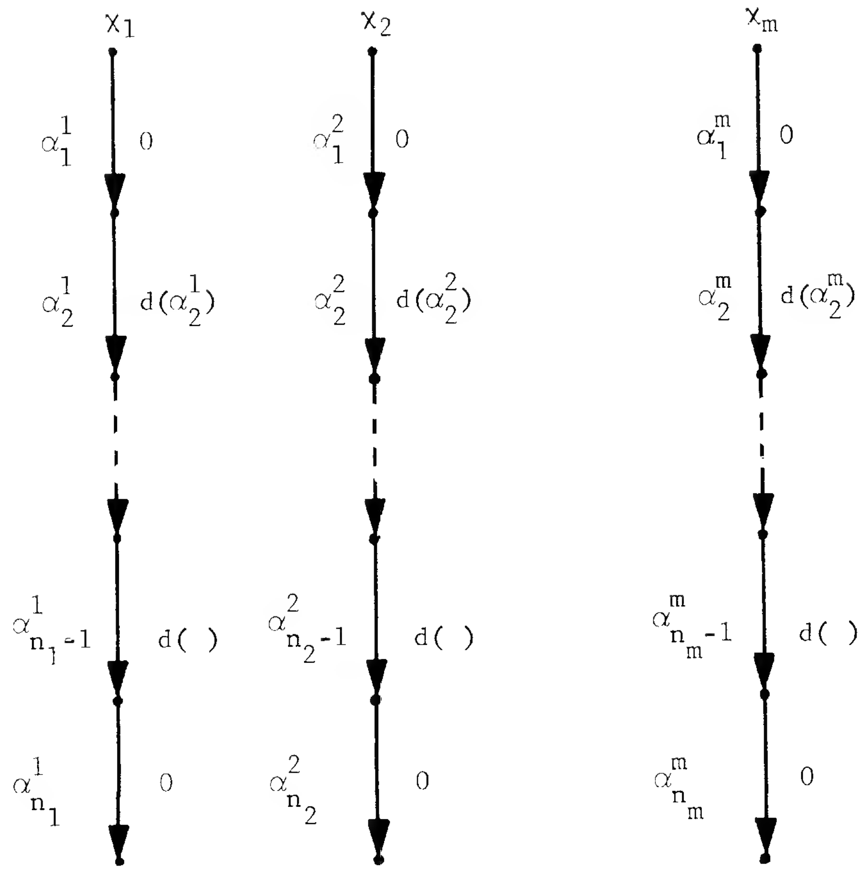
A demand graph is denoted by D with appropriate superscripts when two or more graphs have to be distinguished. The chains of a demand graph will be denoted by χ_i (chi-i) where the suffix is an integer and serves to identify the chain being denoted. In general, there will be m chains so that i assumes values from the set of integers $\{1, 2, 3, \ldots m\}$, which will be denoted by $[1, m]$. The arcs of the demand graph are denoted by their labels, α_j^i, where the superscript i identifies the chain and the subscript j the position of the arc on this chain. The arcs on a chain are numbered in increasing order in the direction of the arrows. The quantity n_i represents the number of arcs on the chain χ_i. Thus j takes values in the set $[1, n_i]$ for arcs on χ_i. Individual arcs are sometimes denoted by α and β. The demand associated with an arc α_j^i will be represented by $d(\alpha_j^i)$. The arrows on the chains will be assumed to be directed downwards, so that "down a chain" means in the direction of the arrows.

Figure 2.2 shows a typical demand graph from the class of Scalar Demand Graphs and illustrates some of the notation.

§2.5 Slices of a Demand Graph

A slice of a demand graph is a set of arcs, one from each chain; the slice is said to intersect the chains in the respective arcs. A slice is thus conceptually similar to a cut-set of the demand graph — it partitions the transitions of a demand graph into those that lie above it and those that lie below it. The transitions that lie above the slice make up the predecessor set of the slice and those that lie below, the successor set of the slice. The initial slice of a demand graph consists of the set of initial arcs and the terminal slice consists of the set of terminal arcs of the graph.

Slices of a demand graph are represented by lower case Greek letters other than α and β — usually γ. The initial slice of a demand graph is denoted by γ_I and the terminal slice by γ_T. The arc from a chain χ_j that goes into a slice γ is represented by $\gamma \square \chi_j$. It is frequently necessary to refer to a slice obtained from another one by a



Capacity = C

Figure 2.2

substitution of arcs. For this purpose a substitution operation on slices is used. The operation is represented as (x/y), and read "substitute arc x for arc y"; arcs x and y must belong to the same chain. Thus $(x/y)\gamma$ represents the slice obtained by replacing arc y by arc x in γ. The operation can be repeated so that expressions of the form $(\alpha'/\alpha)(\beta'/\beta)\gamma$, which means "replace α and β by α' and β', respectively, in γ", are possible. The notation $(\alpha^i_j/\gamma \sqcap \chi_i)\gamma$ represents the slice obtained when the arc $\gamma \sqcap \chi_i$ from the slice γ is replaced by the arc α^i_j. Slices are also represented by a string made up of the labels of the arcs from $\chi_1, \chi_2, \ldots \chi_m$ (in order) that make up the slice. Thus $\alpha^1_1\alpha^2_1 \ldots \alpha^m_1$ is another notation for γ_I and $(\alpha^1_{n_1}/\alpha^1_1)(\alpha^2_{n_2}/\alpha^2_1) \ldots ($ $(\alpha^m_{n_m}/\alpha^m_1)\gamma_I$ is $\alpha^1_{n_1}\alpha^2_{n_2} \ldots \alpha^m_{n_m}$ or γ_T. Figure 2.3 shows several slices; γ_1 is $\alpha^1_1\alpha^2_2$, γ_2 is $\alpha^1_2\alpha^2_1$, and so on. As has been done in Figure 2.3, the arrows on the arcs of demand graphs will be omitted in the figures that follow, unless clarity demands that they be shown.

The slices of a demand graph represent all the states of the system of processes; the arcs composing a slice indicate which phase each process is in. The state of the system is also known as the <u>allocation state</u> of the system since the phases are characterised by steady resource usage. It should be noted that the allocation state is not determined by the set of m demands (and also allocations) of the m processes but rather by the set of m phases — the same set of demands may be encountered for several combinations of phases. The allocation state of the system before any process is initiated is represented by the slice γ_I.

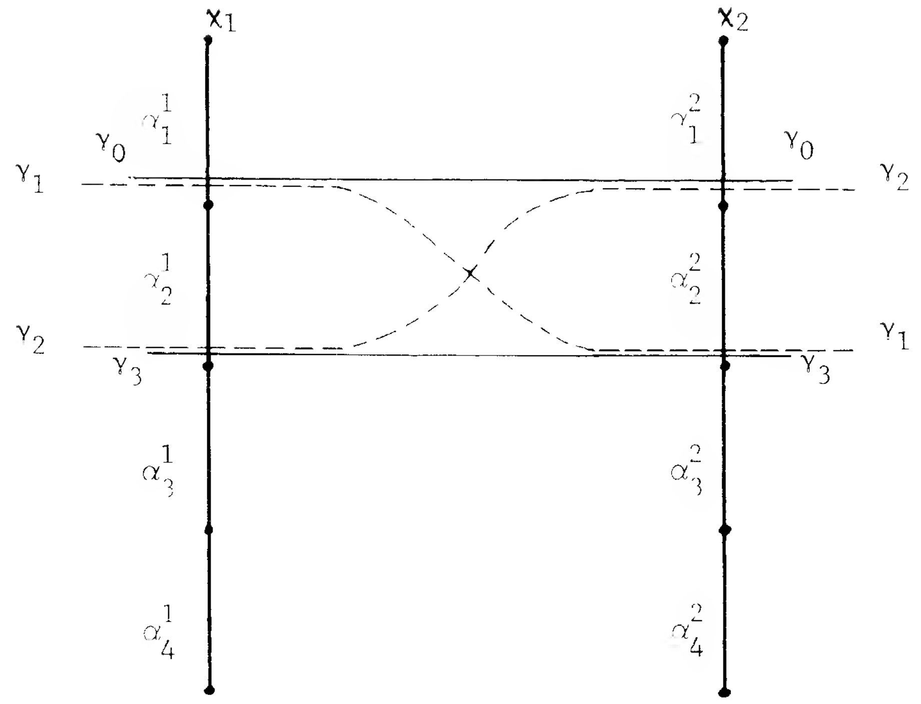

x_1
x_2
α_1^1
α_2^1
α_3^1
α_4^1
α_1^2
α_2^2
α_3^2
α_4^2
γ_0
γ_1
γ_2
γ_3
γ_0
γ_2
γ_1
γ_3

Figure 2.3

As the processes are initiated and progress, the slice representing the current state, i.e. the current slice, moves to lower and lower positions in the demand graph until the state where all the processes have terminated is reached. The last state is represented by the slice γ_T.

§2.6 Relations on the Set of Slices of a Demand Graph

Two relations, viz "earlier than or the same as" and "later than or the same as", can be defined on the set of slices of a demand graph. The relations have the same meaning as their names suggest intuitively. A slice γ_1 is said to be earlier than or the same as a slice γ_2 if the predecessor set of γ_2 includes the predecessor set of γ_1. Predecessor sets are represented by $P(\gamma)$ and successor sets by $S(\gamma)$. The relation "earlier than or the same as" is written "$\preccurlyeq$". Thus $\gamma_1 \preccurlyeq \gamma_2$ if $P(\gamma_2) \supseteq P(\gamma_1)$. Similarly γ_2 is later than or the same as γ_1, written $\gamma_2 \succcurlyeq \gamma_1$, if $S(\gamma_1) \supseteq S(\gamma_2)$, i.e., if the successor set of γ_1 includes that of γ_2. A slice γ_2 is said to be an immediate successor of a slice γ_1 if $\gamma_1 \preccurlyeq \gamma_2$ and if the predecessor set of γ_2 is larger than that of γ_1 by exactly one transition. In general, a slice has m immediate successors. The immediate successor of a slice γ is denoted by $S_i(\gamma)$, where i identifies the chain on which the successor differs from γ in the arc used. In Figure 2.3, γ_1 is the same as $S_2(\gamma_0)$ while γ_3 is $S_1(\gamma_1)$. The strict relations corresponding to "$\preccurlyeq$" and "$\succcurlyeq$" are represented by "$<$" and "$>$", respectively, and are mutually complementary.

The relations "$\leqslant$", "$\geqslant$", "$<$" and "$>$" are also used for arcs with the same meanings, i.e., $\alpha_1 \leqslant \alpha_2$, for instance, means that arc α_1 lies above arc α_2 on some chain. In this connection, the arcs of a chain may be regarded as degenerate slices, i.e., slices of demand graphs that consist of single chains.

§2.7 Partial Orderings and Lattices

A partial ordering is a reflexive† antisymmetric and transitive relation. For example, the ordinary "less than or equal to" relation for integers is a partial ordering. A set with a partial ordering defined on it is a partially ordered set. As explained above, the set of integers is an example of a partially ordered set. A set is said to be totally ordered if every pair of elements is related by the partial ordering relation. The set of integers, for instance, is totally ordered. The set of pairs of integers is only partially ordered — for neither $(2,3) \leq (3,2)$ nor $(3,2) \leq (2,3)$ is true when "$\leq$" is interpreted as requiring that "$\leq$" hold for each pair of corresponding components.

The least upper bound or l.u.b. of a subset, ω, of a partially ordered set, Ω, is the smallest element of Ω that is greater than or equal to every element of ω. Thus the least upper bound of $\{3,5,7\}$ is 7 while that of $\{(3,2), (4,1), (2,5)\}$ is $(4,5)$. The greatest lower bound

†Readers unfamiliar with these terms may wish to consult Birkhoff and MacLane's book [9] or a similar work.

or g.l.b. of a subset, ω, of a partially ordered set, Ω, is the largest element of Ω that is less than or equal to every element of ω.

A lattice is a non-empty partially ordered set, every pair of elements of which has a l.u.b. and a g.l.b. A lattice is said to be a complete lattice if every finite subset of the lattice has a l.u.b. and a g.l.b. It can be shown that every finite lattice, i.e. a lattice with a finite number of elements, is complete. Every finite lattice, therefore, has a least element and a greatest element which are respectively the g.l.b. and l.u.b. of the lattice. The set of pairs of integers from 1 to 10 is a lattice whose least element is (1,1) and greatest element is (10, 10). A lattice is a distributive lattice if the operations of extracting g.l.b.'s and l.u.b.'s distribute over each other. The lattice in the previous example is distributive.

An element a of a lattice is said to cover another element b of the lattice if $b \le a$ but there is no other element x such that $b \le x \le a$. A connected chain in a lattice is a set of elements $x_1, x_2, \ldots x_n$ such that each x_i covers x_{i-1}; the length of such a connected chain is $n - 1$. Two elements x and x' are said to lie on a directed path from x to x' if there exists a connected chain whose first element is x and last element is x'. The length of such a directed path is the length of the connected chain. The Jordan-Holder-Dedekind Theorem for lattices implies that the lengths of all directed paths between a pair of elements of a distributive lattice are equal. For the example in the previous paragraph, the length of any directed path from (2,3) to (5,5) is 5 since 4 elements are required to connect them, e.g. (3,3), (4,3), (5,3), (5,4). Because of this property of distributive lattices, the elements of a distributive lattice can be arranged into ranks — elements at the same distance from the least element of the lattice lie on the same rank.

§2.8 The Lattice of Slices of a Demand Graph

The slices of a demand graph of the kind illustrated in Figure 2.2 form a distributive lattice under the relation "$\preccurlyeq$". The greatest element of the lattice is γ_T while the least element is γ_I. Figure 2.4 shows the lattice of slices of the demand graph of Figure 2.3. The height of the lattice, i.e. the length of a directed path from γ_I to γ_T is $(n_1 - 2) + (n_2 - 2) + \ldots + (n_m - 2)$ or the total number of transitions in the graph.

The l.u.b. of the two slices γ_1 and γ_2 in Figure 2.3 is γ_3 while their g.l.b. is γ_0. This can also be seen in Figure 2.4 where γ_1 is $\alpha^1_1\alpha^2_2$ and γ_2 is $\alpha^2_1\alpha^1_2$, while γ_3 and γ_0 are $\alpha^1_2\alpha^2_2$ and $\alpha^1_1\alpha^2_1$, respectively. In general, the l.u.b. of two slices $\alpha^1_{r_1}\alpha^2_{r_2}\cdots\alpha^m_{r_m}$ and $\alpha^1_{s_1}\alpha^2_{s_2}\ \cdots\ \alpha^m_{s_m}$ is the slice $\alpha^1_{t_1}\alpha^2_{t_2}\ \cdots\ \alpha^m_{t_m}$ where t_i = l.u.b. (r_i, s_i), and similarly for the g.l.b. of two slices.

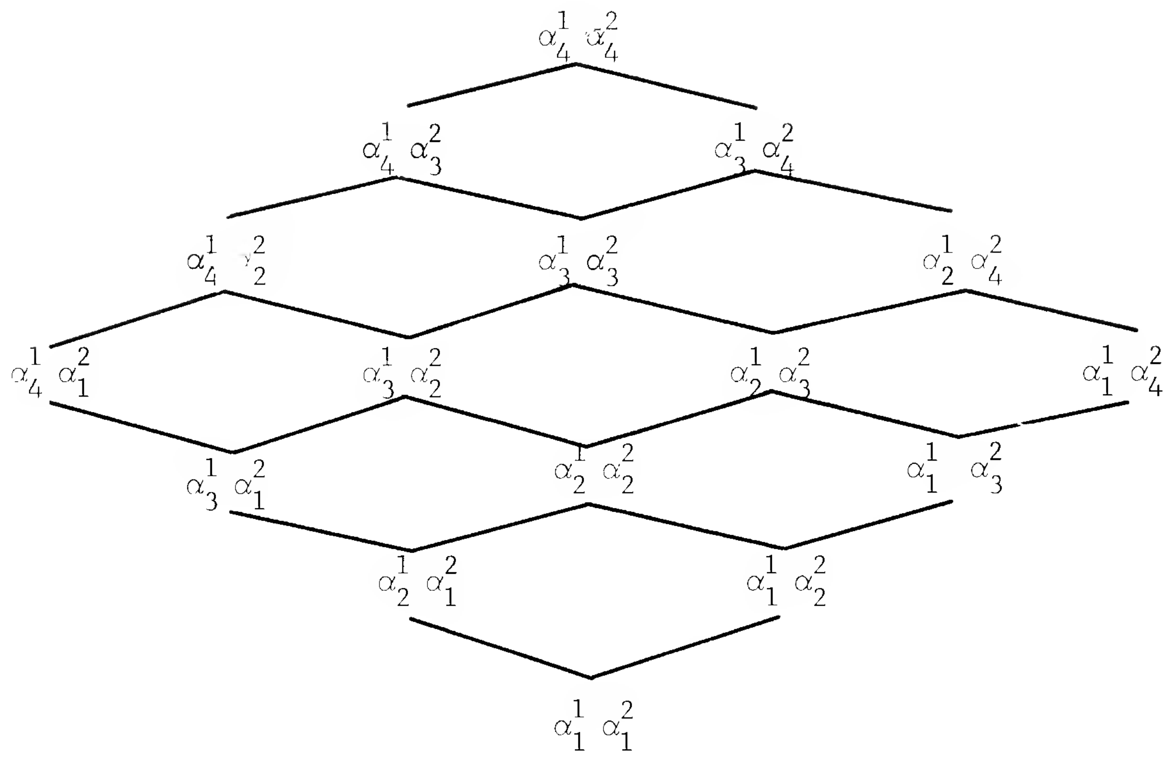

$\alpha_4^1\ \alpha_4^2$
$\alpha_4^1\ \alpha_3^2$
$\alpha_3^1\ \alpha_4^2$
$\alpha_4^1\ \alpha_2^2$
$\alpha_3^1\ \alpha_3^2$
$\alpha_2^1\ \alpha_4^2$
$\alpha_4^1\ \alpha_1^2$
$\alpha_3^1\ \alpha_2^2$
$\alpha_2^1\ \alpha_3^2$
$\alpha_1^1\ \alpha_4^2$
$\alpha_3^1\ \alpha_1^2$
$\alpha_2^1\ \alpha_2^2$
$\alpha_1^1\ \alpha_3^2$
$\alpha_2^1\ \alpha_1^2$
$\alpha_1^1\ \alpha_2^2$
$\alpha_1^1\ \alpha_1^2$

Figure 2.4

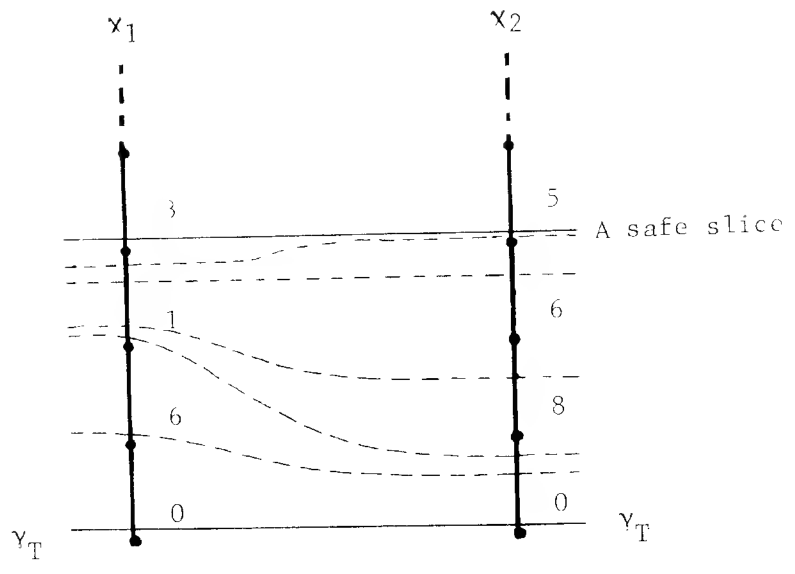
§2.9 Feasibility and Safeness of Slices

A move on a chain χ_i in a demand graph is a function whose domain is the set of all slices which intersect χ_i in a given arc and whose range is the set of immediate successors of these slices on χ_i. A move is thus defined by a pair of typical elements from its domain and range. If a slice, γ, is in the domain of a move, μ, then the corresponding element, γ', in the range of μ is the slice resulting from the application of the move to the slice γ and is represented by $\gamma\mu$. If a move μ, leads from γ_1 to γ_2 then μ is also represented by $\gamma_1 \rightarrow \gamma_2$. Two moves, μ_1 and μ_2, are said to be connected if they can be represented in the form $\gamma_1 \rightarrow \gamma_2$ and $\gamma_2 \rightarrow \gamma_3$, respectively. A macro-move is a sequence of moves, every pair of which is connected. The sequence of slices $\gamma_1\gamma_2\gamma_3 \cdots \gamma_k$ is a connected sequence of slices if the sequence of moves $\gamma_1 \rightarrow \gamma_2$, $\gamma_2 \rightarrow \gamma_3 \ldots$, $\gamma_{k-1} \rightarrow \gamma_k$ is a macro-move. A macro-move from the initial slice, γ_I, of a demand graph to its terminal slice, γ_T, is called a run. A uni-chain macro-move is a macro-move all of whose components are moves on the same chain.

A slice is said to be feasible if the sum of the demands associated with the arcs in it is no greater than C, the capacity associated with the demand graph. A slice that is not feasible is infeasible. A feasible slice of a demand graph is safe if there exists a macro-move from it to the terminal slice of the graph and if the slice resulting from the application of each move in the macro-move is feasible, i.e., if there exists a connected sequence of feasible slices from the slice in question to the terminal slice of the graph. A slice that is not safe is said to be unsafe. Figure 2.5 shows a safe slice[†] and the moves that lead from it to γ_T. In terms of the lattice of slices, a slice γ is safe if there exists a directed path from γ to γ_T, the terminal slice of the graph, that uses only feasible slices.

In terms of the system of processes represented by a demand graph, a feasible slice represents a meaningful allocation state. A feasible slice that lies on a directed path from γ_I which uses only feasible slices represents an attainable allocation state. That a slice is safe means that there exists a schedule for the processes that leads, from the state of the system represented by the slice, to the state in which all the processes have terminated; for each feasible slice resulting from the application of a move to a feasible slice that represents an attainable state, itself represents an attainable state. A slice all of whose immediate successors are infeasible represents a state of deadlock. The slice representing the current state is referred to as the current slice. That the current state is not safe, or is unsafe, implies that every sequence of macro-moves when applied to the current slice eventually leads to a slice all of whose immediate successors are infeasible; there is no schedule for the processes that permits all the processes to complete — deadlock is unavoidable. Because of this association of

[†]It should be noted that if Habermann's analysis were used in this example, the slice marked safe would be declared unsafe. The larger number of slices that can be safe is indicative of the ability to improve resource utilization that the systems discussed here possess.



Capacity = 10

Figure 2.5

§2.10 Representation of Habermann's Systems

It will be recalled from the discussion of Chapter 1 that Habermann studied deadlock avoidance in systems of independent sequential processes in which the only available a priori information about resource usage by processes is that of the maximum amount of each kind of resource that a process uses. Such systems will be known as Habermann systems. As the discussion in this chapter (and Habermann's analysis in [2]) concerns systems with a single type of shared resource, the maximum amount for that resource can be assumed to be available in such a system.

The demand graphs of Figure 2.6a and b represent such systems. In Figure 2.6 $\max_i$ represents the maximum amount of resource that process i ever uses. There are $\max_i$ arcs, in addition to the initial and terminal arcs, on chain χ_i in Figure 2.6b. Figure 2.6b permits representation of allocation states in which a process has been allocated some resource but not the maximum amount it ever needs — this is not possible in Figure 2.6a.

In either of the demand graphs of Figure 2.6, it is clear that a slice is safe if and only if a sequence of uni-chain macro-moves, each of which consists in crossing all the remaining transitions on the chain, can lead from the slice to γ_T by way of feasible slices alone. This is because the demands increase monotonically up to the penultimate arc on each chain. When interpreted this means that a state is

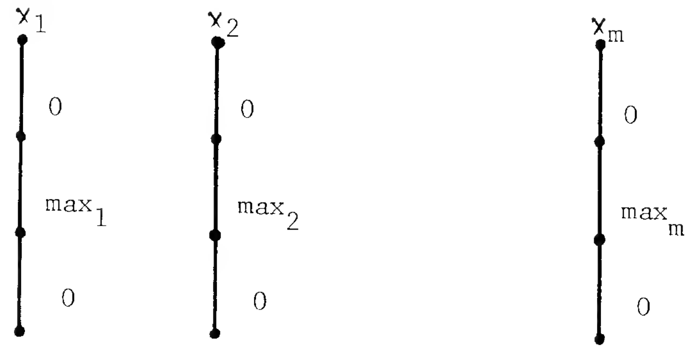


Figure 2.6a

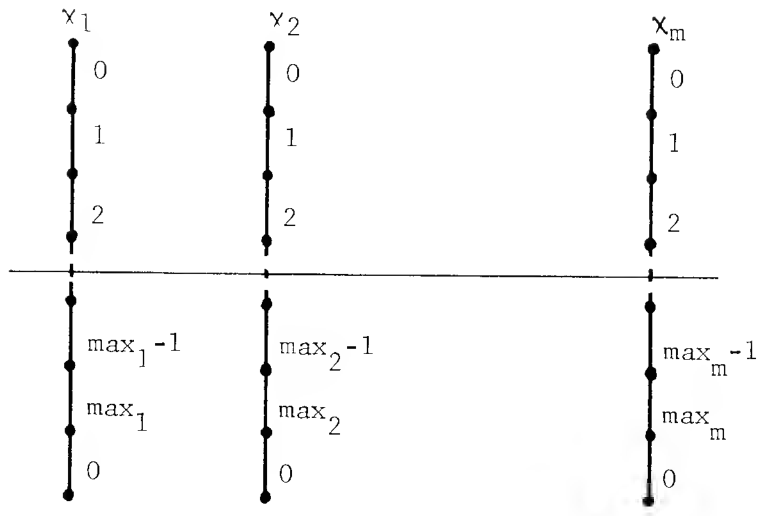


Figure 2.6b

represented by a safe slice if and only if the processes can be scheduled so as to run to completion one at a time (no interleaving of processes). This is exactly Habermann's Theorem 1 in section 2.3 of his thesis [2].

As they stand, neither of the demand graphs of Figure 2.6 really model Habermann systems in all their detail when Habermann's model is interpreted broadly. Firstly, they suggest that allocation to processes is made either all at once (Figure 2.6a) or one server at a time (Figure 2.6b) and this need not be assumed in Habermann systems. However, as phases of processes may last for vanishingly small lengths of time, the representation of Figure 2.6b does not represent a serious distortion. Secondly, after a process has been allocated the maximum amount of resource it ever uses, both the graphs suggest a sudden return en bloc. This behavior is not necessarily shown in Habermann systems either. However, the next section shows that partial return of resources by processes at unknown stages can be represented in the demand graphs for such systems. Thus Habermann's systems are indeed special cases of the systems that can be represented by rectilinear demand graphs.

§2.11 Dynamically Available Resource Usage Information

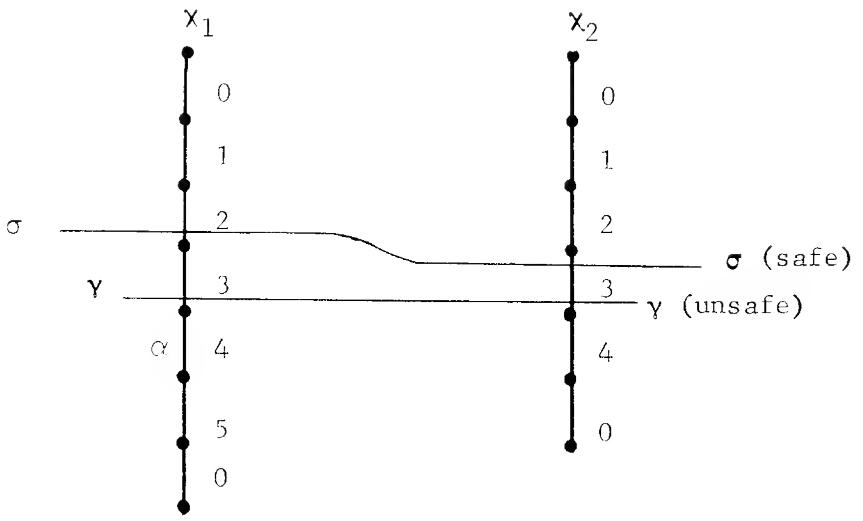
Consider the demand graph of Figure 2.6b. Suppose a slice such as γ were safe. This implies the existence of a sequence of uni-chain macro-moves that lead from γ to γ_T by way of feasible slices and each of which involves crossing all the remaining transitions on a chain.

Consider a segment of a chain that lies entirely below γ and that does not include the terminal arc. If this segment is replaced by another segment, that is of any length whatsoever and the demand on whose arcs does not exceed the largest demand of any arc in the segment removed, then it is clear that the same sequence of macro-moves can still be used. The slice γ is thus safe in spite of this substitution. Figure 2.7 illustrates this for a specific example. For general scalar demand graphs of the kind illustrated in Figure 2.2, if the replacement is restricted to segments consisting of single arcs, then a similar assertion can be made.

One can interpret the discussion of the previous paragraph as implying that any information about future resource usage that becomes available dynamically can be accommodated without deleterious effect if the new information does not contradict an earlier and more conservative estimate. In general, the addition of such information makes safe some states that were unsafe before and thus improves the potential for efficient utilization of resources (see Figure 2.7b).

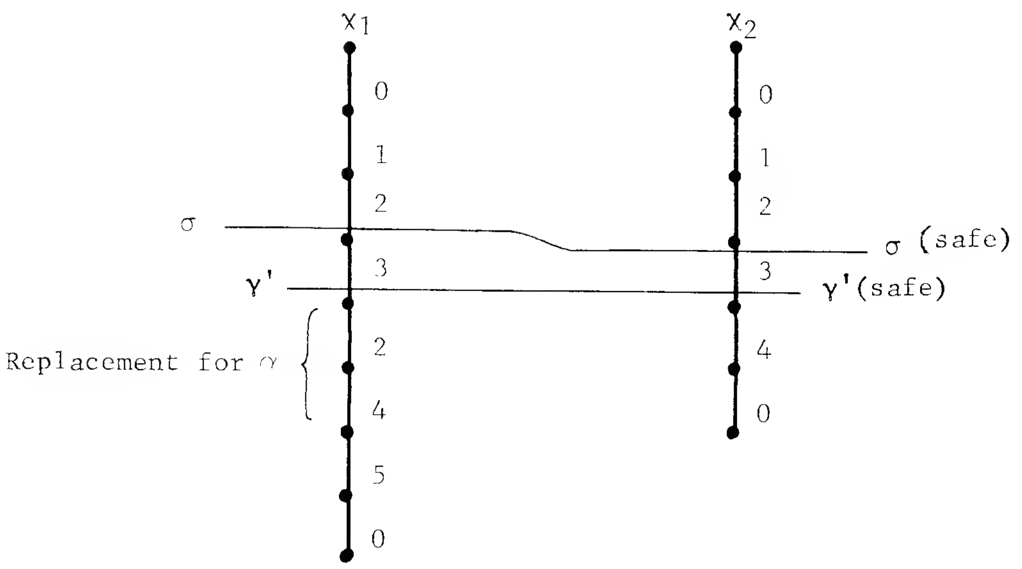
It should be clear, now, that it is possible to use demand graphs to represent systems that exhibit the kind of behavior that Habermann systems can display, i.e., systems that return resources partially.

The discussion of this section shows that demand graphs can be used to represent systems in which additional information about resource usage becomes available during the running of processes.



Capacity = 6

Figure 2.7a



Capacity = 6

Figure 2.7b

§2.12 Safeness Tests

It was indicated in section 2.8 that the avoidance of deadlock requires ensuring that the allocation state is always represented by a safe slice. It is important, therefore, to be able to test a slice for safeness.

One could examine all the slices of a demand graph for feasibility and eliminate those slices that are infeasible from the lattice of slices. Then a slice is safe if a directed path from it to γ_T still exists. By examining every slice for safeness in this manner one could mark all slices that are safe. An allocator desirous of investigating the safeness of a slice, then, need merely determine if it is marked safe. Unfortunately, there are $\prod_{i=1}^{m} n_i$ slices in the lattice while in any run only $1 + \sum_{i=1}^{m} (n_i - 1)$ slices are encountered. Much of the effort in such a scheme is thus wasted. Moreover, if a new chain is added to the graph (corresponding to addition of a process to the system), a similar computation has to be re-done! For these reasons, the safeness tests that are of interest to a resource allocator are incremental tests, i.e., those that test a single slice at a time for safeness — presumably the slice that represents the next state that may become current. Such tests will, in general, attempt to construct a sequence of moves from a test slice to γ_T while ensuring that each move results in a feasible slice.

The next section describes a safeness test in the form of an algorithm for the construction of a sequence of the kind described. An important virtue of this algorithm is that it is non-enumerative, i.e., it does not require the examination of all possible sequences of moves from the slice being tested.

§2.13 The Safeness Algorithm

The slice being tested is assumed to be σ. The slice γ is a variable of the algorithm, as is the set $\{X\}$ which consists of chains of the demand graph.

Step 0: Set γ equal to σ and $\{X\}$ equal to $\{\chi_1, \chi_2, \ldots \chi_m\}$. Go to step 1 if γ is feasible. If γ is infeasible go to step 5.

Step 1: Pick a chain from $\{X\}$ — call it χ_i. Go to step 2.

Step 2: Attempt to construct a uni-chain macro-move down χ_i from γ so that the slice resulting from each component move is feasible. Terminate the macro-move at the first point where a slice — call it γ' — results that satisfies both

$$d(\gamma' \sqcap \chi_i) \leq d(\gamma \sqcap \chi_i)$$

and

$$d(S_i(\gamma') \sqcap \chi_i) \nleq d(\gamma' \sqcap \chi_i)$$

If such a sequence can be constructed go to step 4; if not (i.e., if some move results in an infeasible slice) go to step 3.

<u>Step 3</u>: Delete χ_i from $\{X\}$. If $\{X\}$ is now empty, go to step 5; if not go to step 1.

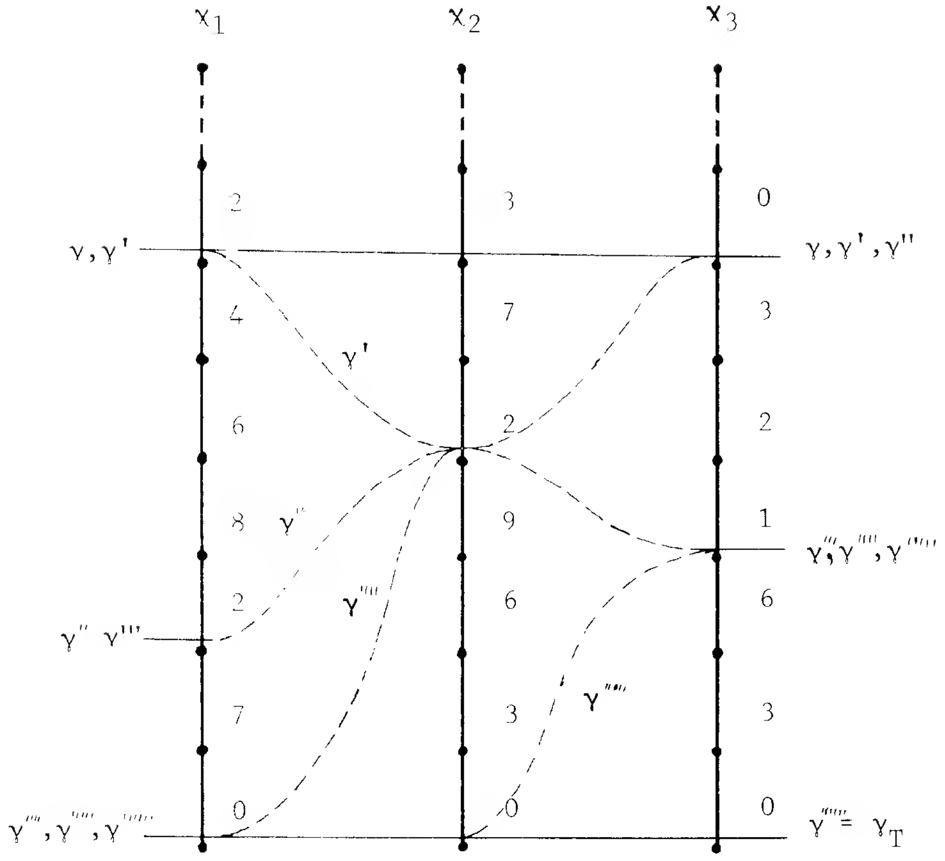
<u>Step 4</u>: If γ' is not γ_T, then replace γ by γ', set $\{X\}$ equal to $\{\chi_1,\chi_2, \ldots \chi_m\}$ and go to step 1. If γ' is γ_T then stop and report success (σ is safe).

<u>Step 5</u>: Stop and report failure (σ is unsafe).

It is clear that when the Safeness Algorithm (called SA for brevity) terminates successfully, σ is safe. Theorem 2.1 below shows that when SA terminates unsuccessfully, σ must be unsafe. An interpretation of the algorithm shows that it seeks the first local minimum of demand that can be found next. When such a local minimum is found, the search is iterated for the new slice and this continues until γ_T is reached. Figure 2.8 shows a sequence of moves constructed by means of the Safeness Algorithm.

The proof of Theorem 2.1 uses the concept of barriers. A <u>barrier</u>, β_i, on a chain, χ_i, <u>with respect to a slice γ</u> is an arc on χ_i that is the first arc below γ for which the predicate $\{(\beta_i/\gamma \sqcap \chi_i)\gamma$ is an infeasible slice$\}$ is true.

<u>THEOREM 2.1</u> A feasible slice, σ, of a demand graph, D, is safe if and only if the Safeness Algorithm terminates successfully when applied to σ and D.



Capacity = 10

Figure 2.8

An examination of SA shows that every uni-chain macro-move in Step 2 of SA leads to a slice γ' which satisfies:

$$d(\gamma' \sqcap \chi_i) \leq d(\alpha) \text{ for all arcs } \alpha \text{ that lie between } \gamma \text{ and } \gamma' \text{ (inclusive) on } \chi_i .$$

Thus it is seen, by a chain of deductions, that γ_0 satisfies

$$d(\gamma_0 \sqcap \chi_i) \leq d(\alpha) \text{ for all arcs } \alpha \text{ that lie between } \sigma \text{ and } \gamma_0 \text{ (inclusive) on } \chi_i$$

for all chains χ_i.

Therefore, in particular,

$$d(\gamma_0 \sqcap \chi_j) < d(\gamma_s \sqcap \chi_j)$$

Therefore, $(\gamma_0 \sqcap \chi_j/\gamma_s \sqcap \chi_j)\gamma_s$ is feasible; for γ_s is feasible.

<u>Case 2</u> $\gamma_0 \sqcap \chi_j < \gamma_s \sqcap \chi_j < \beta_j$

In this case (as explained at the beginning of the proof) too

$$d(\gamma_0 \sqcap \chi_j) < d(\gamma_s \sqcap \chi_j)$$

and so $(\gamma_0 \sqcap \chi_j/\gamma_s \sqcap \chi_j)\gamma_s$ is again a feasible slice by virtue of the feasibility of γ_s.

Thus in either case, one can replace all the arcs in γ_s except β_k by the corresponding arcs of γ_0 and still get a feasible slice. But the resulting slice is $(\beta_k/\gamma_0 \sqcap \beta_k)\gamma_0$ and this is infeasible by assumption! This contradiction implies that γ_0 and hence the sequence Σ cannot exist. Thus, σ must be unsafe.

That σ is safe if SA terminates successfully, follows from the definitions of safeness and successful termination.

Q.E.D.

Theorem 2.1 shows that any macro-move of the kind that leads from σ to a slice γ'''''' which is of the form described in Step 2 of SA can be applied fearlessly to σ in the construction of a feasible sequence of slices from σ to γ_T — backtracking beyond γ'''''' is never required. This leads to Corollary 2.1.1.

<u>COROLLARY</u> <u>2.1.1</u> Let σ be a safe slice of a demand graph D and let σ_i be an immediate successor of σ resulting from a move down a chain χ_i. Let σ_i be feasible and let $\mu_1\mu_2 \ldots \mu_k$ be a macro-move that leads from σ_i to a slice σ_i' by way of feasible slices. Then if

(i) $d(\sigma_i' \sqcap \chi_i) \leq d(\sigma \sqcap \chi_i)$

and (ii) $d(\sigma_i' \sqcap \chi_j) \leq d(\sigma \sqcap \chi_j)$ for all chains χ_j on which σ_i' and σ differ in the arcs chosen

then σ_i is a safe slice.

The corollary follows since $\mu_1\mu_2 \ldots \mu_k$ is a macro-move of the kind described in the paragraph above Corollary 2.1.1.

If the macro-move $\mu_1\mu_2 \ldots \mu_k$ in Corollary 2.1.1 is a uni-chain macro-move down χ_i then the test is simplified considerably. Thus it should be profitable to look for such a uni-chain macro-move. In any case, as long as $\sigma_i' \prec \gamma_T$, some labour is saved.

COROLLARY 2.1.2 Let σ be a safe slice of a demand graph and let σ_i be $S_i(\sigma)$. Then if $d(\sigma_i \sqcap \chi_i) \leq d(\sigma \sqcap \chi_i)$ then σ_i is safe.

This corollary follows from Corollary 2.1.1 since the move $\sigma \to \sigma_i$ is itself the macro-move that satisfies the conditions of that corollary.

Theorem 2.1 and its corollaries point out that the Safeness Algorithm, shortened as suggested in Corollary 2.1.1 and 2.1.2 whenever possible, provides a simple test for the use of an allocator of resources.

It should be pointed out that a sequence of feasible slices from σ to γ_T which is constructed by the Safeness Algorithm does not represent the actual schedule or order in which processes will be allowed to proceed (by the allocator). The actual order may be quite different, being determined by actual requests from the processes, to be permitted to proceed to their respective next phases of activity, together with considerations of safeness of the slices corresponding to the allocation states of the system which would result if the requests were granted. This is the incremental aspect of tests that was referred to earlier. It is this incremental approach that permits dynamic increase of the number of processes in the system as well as the dynamic changes, in the demand graphs of processes already in the system, that were described in section 2.11.

[illegible]

[illegible]

[illegible]

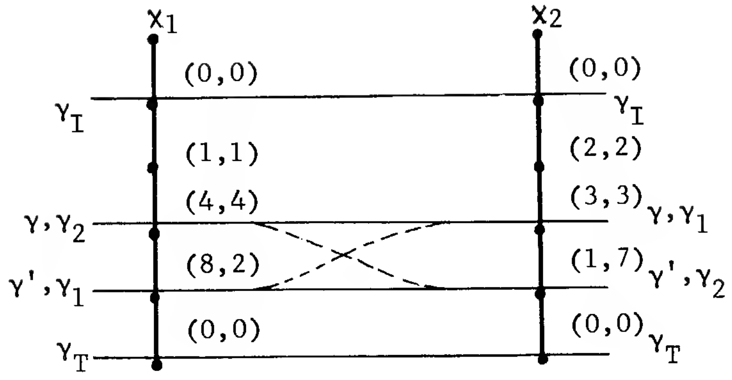
[illegible]

§3.1 Rectilinear Vector Demand Graphs

The discussion and analysis of Chapter 2 dealt with the representation and analysis of systems in which a single type of resource is shared from a pool in an unpre-emptable manner. As the construction analogue of Chapter 1 illustrates, however, there are many systems in which more than one type of resource are so used. An extension of the analysis of deadlocks to systems with multiple resource types, or multi-resource systems, for brevity, is therefore of interest and is the goal of this chapter.

Sections 3.11 and 3.12 illustrate how the sharing of locked data bases in computer systems and Job Shop Scheduling can be analysed using the representation and analysis developed in Sections 3.2 to 3.10.

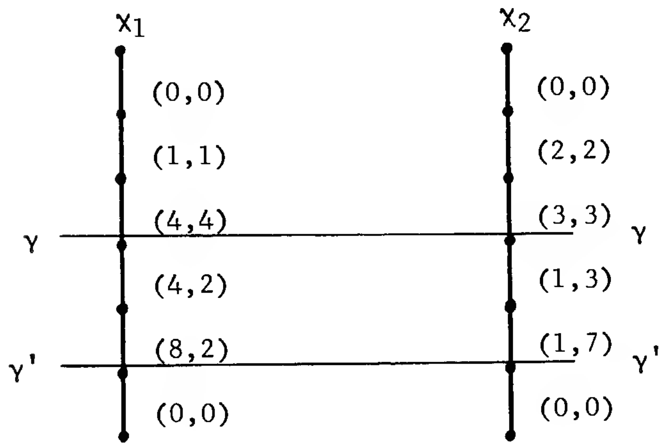
Multi-resource systems can be represented by Rectilinear Vector Demand Graphs, or Vector Demand Graphs for brevity, which are structurally identical to Rectilinear Scalar Demand Graphs except that Δ for such graphs is the set of n tuples of non-negative integers for some specified n. The arcs of Vector Demand Graphs, therefore, have n-tuples or vectors of demand instead of scalar demands associated with themselves. The vectors of demand are represented by $\underline{d}(\alpha)$ to emphasize this difference. As before, convention dictates that the initial and terminal arcs of each chain have zero demand, i.e. (0, 0, ... 0), associated with them. Figure 3.1a illustrates such a demand graph.



Capacity = (10,10)

Demand Graph D

Figure 3.1a



Capacity = (10,10)

Demand Graph D Transformed

Figure 3.1b

The terminology of Chapter 2 carries over mutatis mutandis — the qualification refers to the replacement of a scalar capacity, C, by a vector capacity, $\underline{C}$, and of scalar inequality by vector inequality.

§3.2 A Transformation for Vector Demand Graphs

A peculiar phenomenon appears in Vector Demand Graphs in connection with safeness. It will be noticed in Figure 3.1a that the slice of D marked γ is unsafe because both γ_1 and γ_2, the two slices which are immediate successor slices of γ, are infeasible. However, γ' is feasible. Moreover, in the system which is represented by D, the state represented by γ' can be attained just after that represented by γ; for it merely corresponds to responding (favourably and) simultaneously to the requests from two users to be permitted to proceed to their respective next phases of activity. Thus the state corresponding to γ should be safe.

To be able to make γ safe would require changing the definition of a move to permit crossing of several transitions in a move.

However, representation of such simultaneous or multiple moves in the lattice of slices requires addition of a large number of paths to the lattice; for at each node of the lattice there would be, in general,

$$2^r = \sum_{1}^{r} {}^rC_k \qquad 1 \le r \le m$$

possible successors, viz the slices that can be reached by multiple moves that involve crossing up to r transitions simultaneously. Moreover, an algorithm such as the Safeness Algorithm has to examine these 2^r possible successor slices one by one until its test is satisfied. This increases immensely the amount of work involved in examining the safeness of a slice.

Fortunately, a transformation of the demand graphs (as illustrated in Figures 3a and 3b) produces a demand graph in which every slice of the original demand graph that was safe, when multiple moves are permissible, is safe when only single moves are permissible. The transformation operates on pairs of adjacent arcs, typified by α_1 and α_2 say, on each chain. Whenever $\underline{d}(\alpha_1) \nleq \underline{d}(\alpha_2)$ and $\underline{d}(\alpha_2) \nleq \underline{d}(\alpha_1)$ (where "$\leq$" is understood in the usual vector sense of each component of the Left Hand vector being less than or equal to the corresponding component of the Right Hand vector), an arc α_1' is introduced between α_1 and α_2 with a demand which is the greatest lower bound of the two vectors $\underline{d}(\alpha_1)$ and $\underline{d}(\alpha_2)$. Thus $\alpha_1 < \alpha_1' < \alpha_2$ and $\underline{d}(\alpha_1) > \underline{d}(\alpha_2) < \underline{d}(\alpha_3)$. It should be clear that these arcs which are inserted provide a sequence of single-transition moves between every pair of slices of the type γ and γ' in Figure 3.1, with only feasible slices resulting from the moves.

As the transformation described above is vital to the accuracy of representation and analysis of multi-resource systems by demand graphs, it will be presumed that such a transformation is carried out before any algorithms or tests are applied to demand graphs. However, the transformation is not crucial to the analysis that is presented.

§3.3 The Modified Safeness Algorithm

As in Chapter 2, an incremental algorithm for the determination of the safeness of a slice is desirable. It is tempting to try and use the Safeness Algorithm of Chapter 2 with a vector comparison in step 2, instead of a scalar one. That step would read:

> Step 2: Attempt to construct a uni-chain macro-move down χ_i so that the slice resulting from each component move is feasible. Terminate the macro-move at the first point where a slice γ' is reached that satisfies:
>
> $\underline{d}(\gamma' \mathbin{\Box} \chi_i) \leq \underline{d}(\gamma \mathbin{\Box} \chi_i)$
>
> and $\underline{d}(S_i(\gamma') \mathbin{\Box} \chi_i) \nleq \underline{d}(\gamma' \mathbin{\Box} \chi_i)$
>
> where "$\leq$" is interpreted as holding for all components simultaneously.
>
> If the attempt is successful, go to step 4; if not (i.e., if some move results in an infeasible slice), go to step 3.

Consider Figure 3.2a. Were one to apply the algorithm as modified above to slice γ, one would get to γ'', by way of γ', and discover that no moves from γ'' result in feasible slices. Unfortunately, the failure of the algorithm at γ'' does not imply (as it would in the case of the Safeness Algorithm for Scalar Demand Graphs) that γ is unsafe. For

the sequence of slices $\gamma - \gamma^{*} - \gamma^{**} - \gamma^{***}$, in that figure, shows part of a full sequence of feasible slices from γ to γ_T.

The slice γ^{*} suggests that avoidance of erroneous moves requires changing the condition to be satisifed by γ' in step 2 of the algorithm above to

$$\underline{d}(\gamma' \square \chi_i) \le \underline{d}(\delta \square \chi_i) \quad \text{for all slices } \delta \text{ that lie between } \gamma \text{ and } \gamma' \text{ (inclusive)}$$

$$\text{and} \quad \underline{d}(S_i(\gamma') \square \chi_i) \nleq \underline{d}(\gamma' \square \chi_i)$$

Corollary 3.1.2 of Theorem 3.1 below proves the validity of this conclusion. The Safeness Algorithm of Chapter 2 with the condition in Step 2 replaced according to this suggestion will be referred to as the Modified Safeness Algorithm.

A few definitions are required for the precise statement of the result of Theorem 3.1, and these follow.

§3.4 The Prefix Property

The set of extensions, $E_D(\gamma)$, of a demand graph, D, with respect to a slice, γ, of D is the set of all demand graphs which are identical to D up to γ, have the same capacity associated with themselves as D, and have at least one arc below γ on each chain. The demands associated with the arcs below γ are not constrained except by the definition of a Vector Demand Graph. A member of $E_D(\gamma)$ is called an extension of the demand graph, D, with respect to the slice γ. Figure 3.2b shows an extension of the demand graph of Figure 3.2a with respect to γ^*. Extensions of a demand graph with respect to a slice represent possible continuations of the demand graph beyond that slice.

A feasible slice, γ, of a demand graph, D, which can be reached by a connected sequence of feasible slices from an earlier slice, σ, is said to possess the prefix property with respect to the slice, σ, if in all extensions of D with respect to γ in which σ is safe, γ is safe too. Let P be the prefix relation "possesses the prefix property with respect to". Then P is clearly transitive, so that $\sigma P \gamma$ and $\gamma P \gamma'$ implies $\sigma P \gamma'$. This transitivity is very valuable and will be utilized extensively.

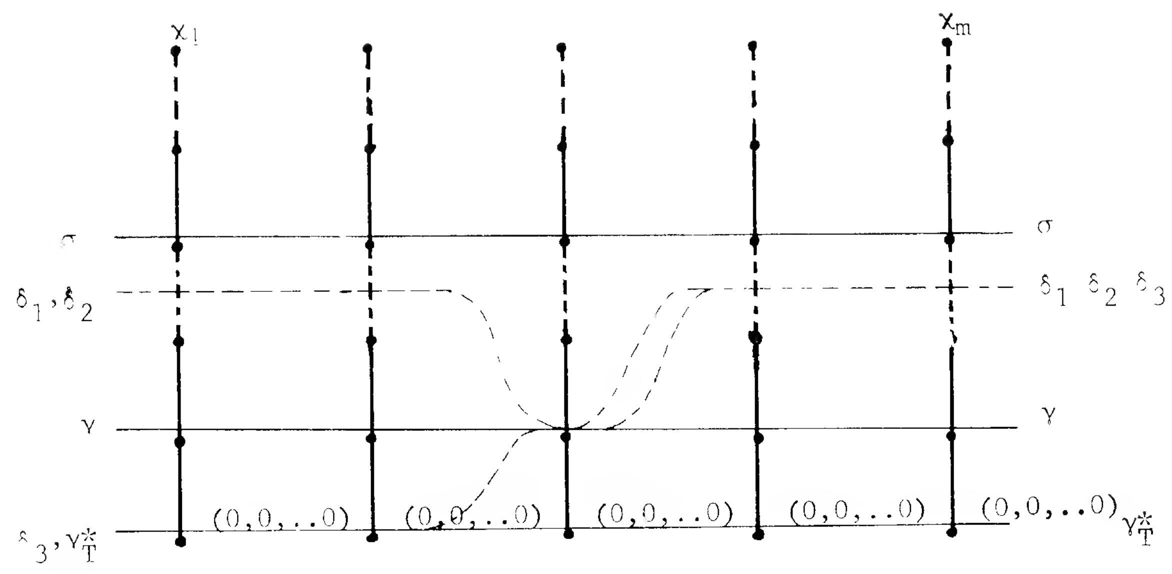
§3.5 Necessary and Sufficient Conditions for the Prefix Property

In terms of the prefix property, it will be seen that for Scalar Demand Graphs, the condition of Step 2 of the Safeness Algorithm (see Chapter 2) is sufficient for possession of the prefix property by a slice, i.e. by γ' with respect to γ. Lemmas 3.1 and 3.2 state necessary and sufficient conditions, respectively, for a slice of a Vector Demand Graph to possess the prefix property with respect to another slice. In these lemmas and in the rest of this thesis, the term "accessible" means "can be reached by a connected sequence of feasible slices". Also, "$\underline{d}$ (a slice)" is the concise notation for the sum of the demands on the arcs in the slice — the object in parentheses may be only a part (subset) of a slice and then the notation stands for the sum of the demands on the arcs in that part of the slice. The term "a move <u>fits</u> a slice <u>feasibly</u>" in the proofs of these lemmas means that the slice, γ, is feasible and in the domain of the move, μ, and $\gamma\mu$ is a feasible slice. A macro-move fits a slice feasibly if each component move fits feasibly the slice resulting after the previous component moves have been applied.

<u>Note</u>: The case in which a slice of a demand graph passes through only one arc that has a non-zero demand is a degenerate one. That is to say, every such slice possesses the prefix property with respect to any earlier slice from which it is accessible; for one process-at-a-time completion is possible, as the demand on each arc of the demand graph does not exceed the capacity of the graph. For this reason that case has been excluded from consideration in Lemmas 3.1 to 3.3 and in Theorem 3.1.

<u>LEMMA 3.1</u> Let D be a vector demand graph and let γ be a feasible slice of D that contains at least two arcs having non-zero demands. Further, let σ be a feasible slice of D from which γ is accessible. Let D^* be the extension of D with respect to γ defined by Figure 3.3 and δ_1 be any slice of D^* that is of the form F_1 defined below. Then the slice γ possesses the prefix property with respect to σ only if whenever the slice δ_1 is accessible from σ, the slice γ is not accessible from δ_1.

<u>Form F_1</u> A slice, δ_1, of this form satisfies the following conditions:



The Extension D*

<u>Figure 3.3</u>

(i) $\sigma \preccurlyeq \delta_1 \prec \gamma$

(ii) δ_1 and γ share at least one arc that has a non-zero demand.

(iii) $\underline{d}(\gamma) \nleq \underline{d}(\delta_1)$

PROOF: Suppose that the condition is violated, i.e. a slice δ_1 of the form F_1 is accessible from σ and γ is accessible from δ_1. Let χ_j be the chain on which the arc common to γ and δ_1 lies.

Consider the extension D' of D with respect to γ that is defined by Figure 3.4 (the chains have been rearranged for drafting convenience).

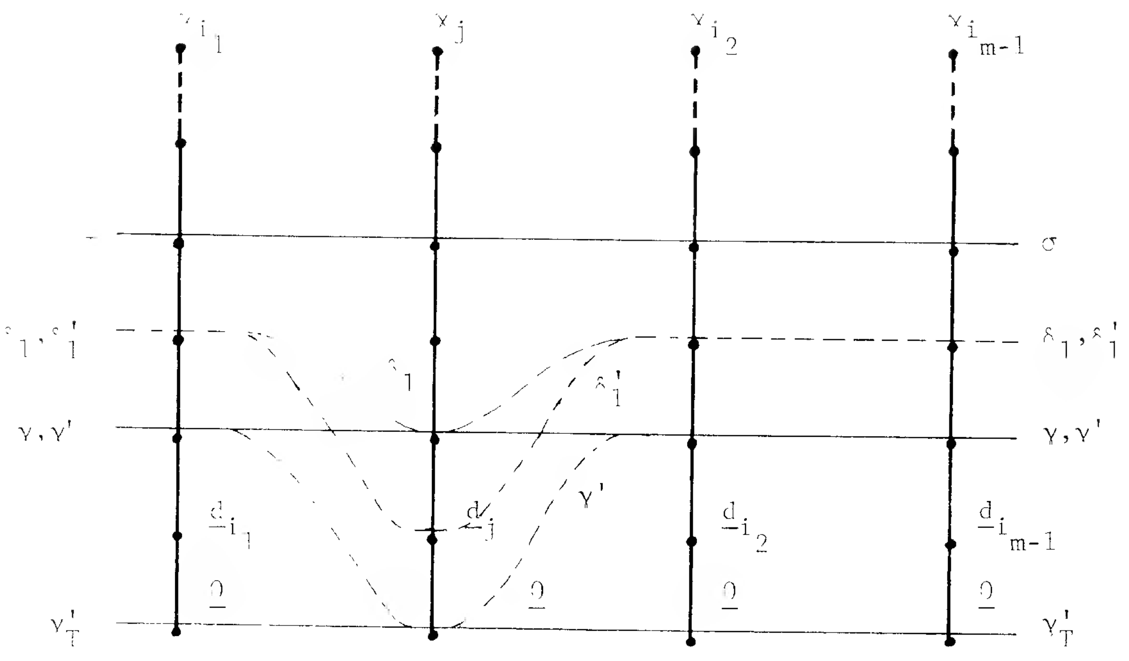
The slice γ is not safe in D' as the values for the demands, d_{i_k} have been chosen so that the only slice later than γ from which γ'_T, the terminal slice of D', is accessible is γ' and γ' is not accessible from γ because $\underline{d}(\gamma) \nleq \underline{d}(\delta_1)$.

However, δ'_1, is clearly accessible from δ_1 and has a smaller demand on each chain than δ_1, i.e.,

$$\underline{d}(\delta'_1 \sqcap \chi_i) \leq \underline{d}(\delta_1 \sqcap \chi_i) \quad \text{for all chains } \chi_i.$$

Thus, since γ is accessible from δ_1, γ' must be accessible from δ'_1.

Now it is clear from the figure that γ' is safe in D'. Consequently, the sequence of macro-moves $\delta \to \delta'$, $\delta' \to \gamma'$ and $\gamma' \to \gamma_T$ is one which produces a connected sequence of feasible slices from δ to γ_T. Thus δ is safe in D' and, consequently, σ is safe in D'.



$$\underline{d}_j = \underline{C} - \sum_{\substack{1 \\ i \neq j}}^{m} \underline{d}(\delta_1 \sqcap \chi_i)$$

$$\underline{d}_{i_k} = \underline{C} - [\,\underline{d}(\gamma') - \underline{d}(\gamma' \sqcap \chi_{i_k})] \qquad k \in [1, m-1],\ i_k \in [\chi_1, \ldots \chi_m]$$

$$= \underline{C} - [\,\underline{d}(\gamma) - \underline{d}(\gamma \sqcap \chi_{i_k}) - \underline{d}(\gamma \sqcap \chi_j)]$$

$$> \underline{C} - [\,\underline{d}(\gamma) - \underline{d}(\gamma \sqcap \chi_{i_k})]$$

The Extension D'

<u>Figure 3.4</u>

However, γ is not safe in D'. Thus γ cannot possess the prefix property with respect to σ.

Q.E.D.

An immediate consequence of Lemma 3.1 is Lemma 3.2

LEMMA 3.2 Let D be a vector demand graph and let γ be a feasible slice of D that contains a least two arcs having non-zero demands. Further, let σ be a feasible slice of D from which γ is accessible. Let D^* be the extension of D with respect to γ defined by Figure 3.3 and δ_2 be any slice of D^* of the form F_2, which is defined below. Then the slice γ possesses the prefix property with respect to σ only if each such δ_2 is inaccessible from σ.

Form F_2 A slice δ_2 of this form satisifes the following conditions:

(i) $\sigma \preccurlyeq \delta_2 \prec \gamma$

(ii) δ_2 and γ share at least one arc that has a non-zero demand.

(iii) $\underline{d}(\gamma) \nleq \underline{d}(\delta_2)$

(iv) $\underline{d}(\delta_2 \boxtimes \chi_i) \leq \underline{d}(\rho \boxtimes \chi_i)$ for all slices, ρ, which lie between σ and δ_2 (inclusive) and for all chains, χ_i.

<u>PROOF</u>: It need merely be shown that condition (iv) in Form F_2 implies that γ is accessible from δ_2, for then the result follows from Lemma 3.1.

Since γ is accessible from σ, there exists a sequence, M, of moves from σ to γ, say it is $\mu_1\mu_2 \cdots \mu_q$. Then $\sigma\mu_1\mu_2 \cdots \mu_q = \gamma$. Also, the slice resulting from the application of each move is feasible, i.e., each μ_i fits the slice $\sigma\mu_1\mu_2 \cdots \mu_{i-1}$ feasibly. Let μ_ℓ be the first move in M that has the property that

$$\sigma\mu_1\mu_2 \cdots \mu_{\ell-1} \preccurlyeq \delta_2$$

but $$\sigma\mu_1\mu_2 \cdots \mu_\ell \not\preccurlyeq \delta_2$$

i.e., μ_ℓ is the first move to cross δ_2. Then, by virtue of condition (iv) in the definition of form F_2,

$$\underline{d}(\delta_2) \le \underline{d}(\sigma\mu_1\mu_2 \cdots \mu_{\ell-1})$$

Thus, μ_ℓ fits δ_2 feasibly.

Similarly, $\mu_{\ell+1}$ fits $\delta_2\mu_\ell$ feasibly, and so on up to μ_p where μ_p is the first move to cross δ_2 completely,

i.e. $$\sigma\mu_1\mu_2 \cdots \mu_{p-1} \not\succ \delta_2$$

but $$\sigma\mu_1\mu_2 \cdots \mu_p \succ \delta_2$$

At this point, $\sigma\mu_1\mu_2 \cdots \mu_p = \delta_2\mu_\ell\mu_{\ell+1} \cdots \mu_p$ and, consequently, the macro-move $\mu_{p+1} \cdots \mu_q$ fits $\sigma\mu_1\mu_2 \cdots \mu_p$ feasibly.

Thus $\mu_\ell \cdots \mu_q$ is a macro-move that fits δ_2 feasibly and $\delta_2\mu_\ell \cdots \mu_q = \gamma$. Thus γ is accessible from δ_2.

[Some moves, μ_m, produce no apparent effect

$$\text{i.e.}\quad \delta_2\mu_\ell \cdots \mu_m = \delta_2\mu_\ell \cdots \mu_{m-1}$$

These moves are those that produce an immediate successor on a chain that still intersects the chain at or above δ_2. These moves can be ignored.]

Q.E.D.

<u>LEMMA 3.3</u> Let D be a vector demand graph and γ be a feasible slice of D that contains at least two arcs that have non-zero demands. Further, let σ be a feasible slice of D from which γ is accessible. Let D^* be the extension of D with respect to γ defined by Figure 3.3 and δ_3 be any slice of D^* that is of the form F_3, which is defined below. Then γ possesses the prefix property with respect to σ if whenever δ_3 is accessible from σ, γ_T^*, the terminal slice of D^*, is not accessible from δ_3.

<u>Form F_3</u> A slice δ_3, of this form, satisfies the following conditions:

(i) $\sigma \preccurlyeq \delta_3$

(ii) Either δ_3 and γ share at least one arc that nas non-zero demand, or δ_3 includes at least one terminal arc of D^*.

$$\sigma\mu_1\mu_2 \cdots \mu_{p-1} \not\geq \gamma$$

$$\sigma\mu_1\mu_2 \cdots \mu_p \geq \gamma$$

Moreover,

$$\sigma\mu_1\mu_2 \cdots \mu_p = \gamma\mu_\ell\mu_{\ell+1} \cdots \mu_p$$

so that the macro-move $\mu_{p+1} \cdots \mu_q$ fits $\gamma\mu_\ell\mu_{\ell+1} \cdots \mu_p$ feasibly.

Thus $\mu_\ell\mu_{\ell+1} \cdots \mu_q$ is a macro-move from γ to γ'_T that has the property that each μ_m fits $\gamma\mu_\ell\mu_{\ell+1} \cdots \mu_{m-1}$ feasibly. Thus γ is safe.

Therefore, γ possesses the prefix property with respect to σ.

Q.E.D.

An immediate consequence of Lemma 3.3 is Theorem 3.1.

THEOREM 3.1 Let D be a vector demand graph and γ be a feasible slice of D that includes at least two arcs that have non-zero demands. Further, let σ be a feasible slice of D from which γ is accessible. Then γ possesses the prefix property with respect to σ if

$\underline{d}(\gamma \sqcap \chi_i) \leq \underline{d}(\rho \sqcap \chi_i)$ for all slices, ρ, that lie between σ and γ (inclusive) and for all chains, χ_i

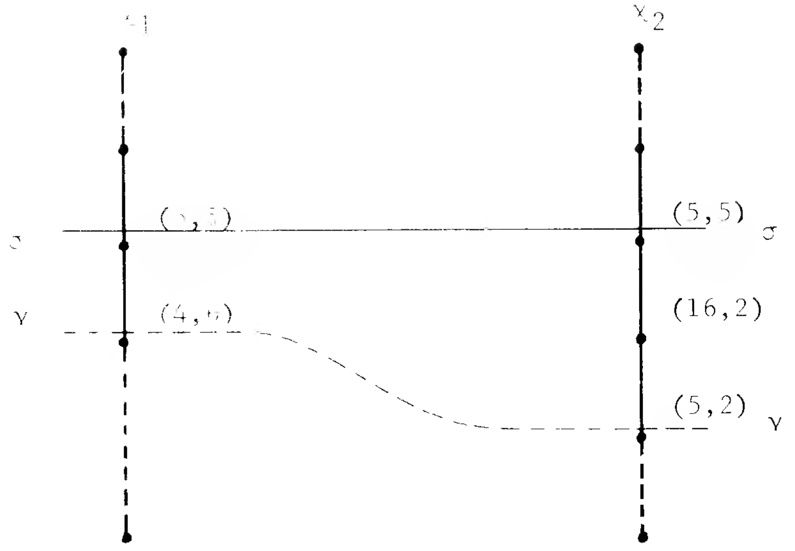
PROOF: From the condition of the theorem it follows that condition (iii) of Form F_3 cannot be met by any slice satisfying conditions (i) and (ii) of that Form. The result, therefore, follows from Lemma 3.3.

The results proved above have little intuitive meaning and their principal use is in proving Theorem 3.3 later. The reader should satisfy himself that the necessary and sufficient conditions in Lemmas 3.1 and 3.3 are compatible. Figure 3.5 shows a slice, γ, which possesses the prefix property with respect to another slice, σ, even though the conditions of Theorem 3.1 are violated — the conditions of Lemma 3.3 are met, however. Theorem 3.1 provides the basis for the Basic Algorithm, which is presented later.

§3.6 Inadequacies of the Modified Safeness Algorithm

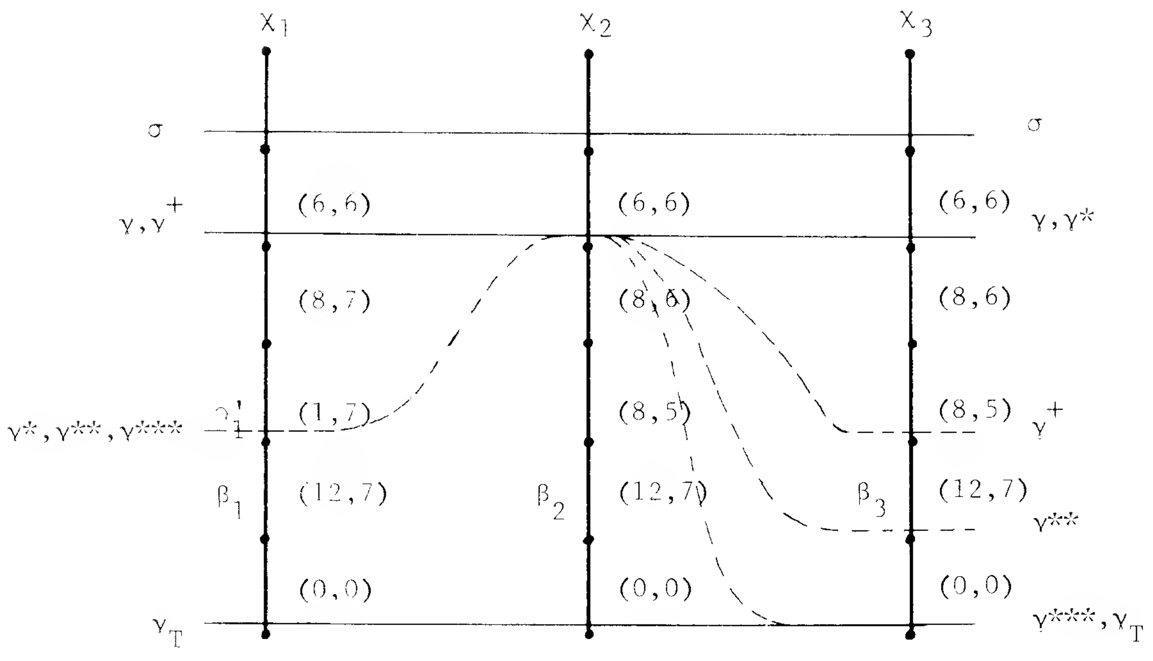
Theorem 3.1, stated above, shows that the slices, γ', produced in Step 2 of both the Safeness Algorithm of Chapter 2 and the Modified Safeness Algorithm possess the prefix property with respect to the slices γ.

The prefix property states that partial sequences of feasible slices possess extensions that lead to the terminal slice. Theorem 2.1 showed that, in addition to producing slices with the prefix property, the Safeness Algorithm of Chapter 2 was always able to construct the extension. Unfortunately, such is not the case for the Modified Safeness Algorithm, and Figure 3.6 illustrates this. In that figure, the Modified Safeness Algorithm fails at γ even though there is an extension, viz $\gamma - \gamma^{*} - \gamma^{**} - \gamma^{***} \ldots \gamma_T$, of the sequence $\sigma - \gamma$. The Modified Safeness Algorithm thus needs to be augmented by an algorithm that constructs such an extension when the former is unable to — the Crutch Algorithm given below is such an algorithm.



Capacity = (20,20)

Figure 3.5



Capacity = (20,20)

Figure 3.6

The example of Figure 3.6 shows that the reason that an extension of the sequence $\sigma - \gamma$ exists, even when the Modified Safeness Algorithm is unable to find one, is that the demand on the arc marked α_1' is sufficiently low in a crucial component, viz the first one, to enable a macro-move down χ_3 to its terminal arc to fit feasibly in spite of the fact that the demand on α_1' is not vectorially less than that on $\gamma \sqcap \chi_1$. An arc such as α_1' is called a crutch for the obvious reason. An arc, α_i, on a chain, χ_i, of a demand graph, D, is said to be <u>a crutch with respect to a slice</u>, γ, of D if the following relation is satisfied:

$$\underline{d}(\alpha_i) \not> \underline{d}(\gamma \sqcap \chi_i)$$

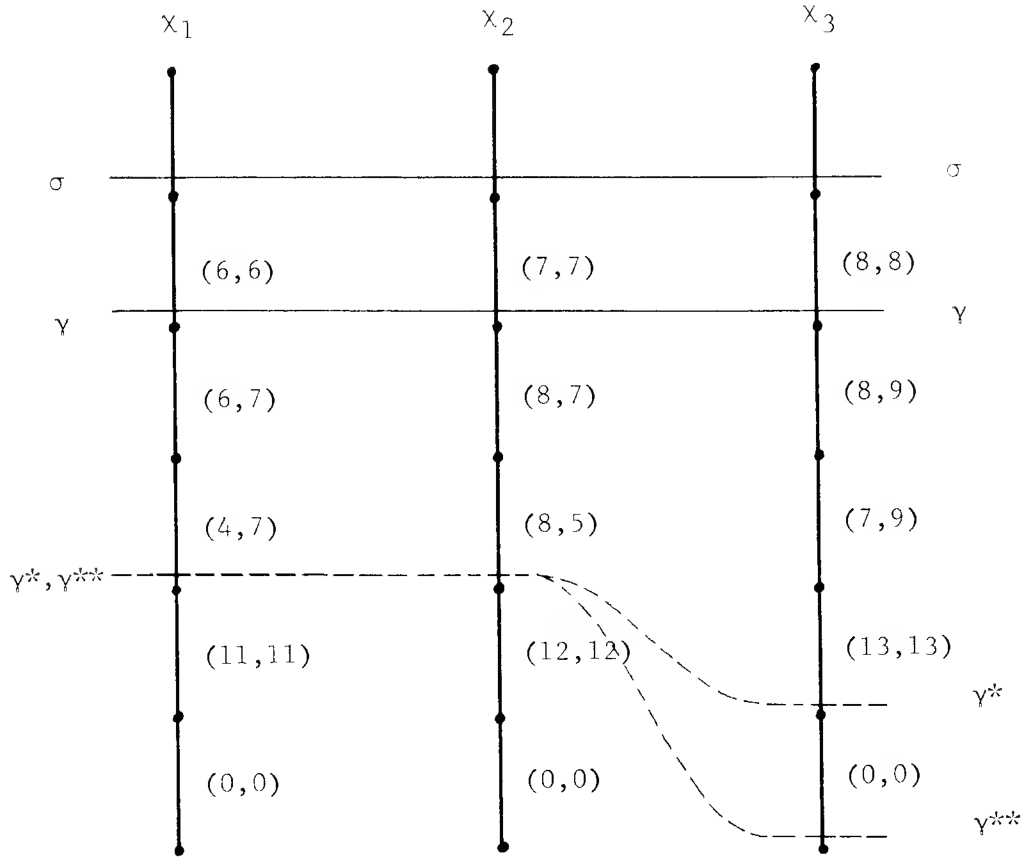
The example in Figure 3.6 also points out that the Modified Safeness Algorithm fails at a slice, γ, of a demand graph when moves down each chain result (eventually) in an infeasible slice before the conditon in Step 2 of that algorithm is satisfied. The arcs on the chains which correspond to these infeasible slices are thus barriers (see the arcs marked β_1, β_2 and β_3 in Figure 3.6 for instance). An arc, β_i, on a chain, χ_i, is said to be <u>a barrier with respect to a slice</u>, γ, which lies above it, if β_i is the first arc on χ_i after $\gamma \sqcap \chi_i$ such that the slice $(\beta_i/\gamma \sqcap \chi_i)\gamma$ is infeasible. When the Modified Safeness Algorithm fails at a slice, γ, then a barrier with respect to γ exists on each chain of the demand graph.

The role of the augmentative Crutch Algorithm can now be explained. When the Modified Safeness Algorithm fails while testing a slice σ for safeness, there exist barriers, β_i on χ_i, with respect to the last slice possessing the prefix property with respect to the slice σ. If no crutches with respect to γ lie between γ and the β's, then extension of the sequence $\sigma \ldots \gamma$ to γ_T is not possible and σ is unsafe. When such crutches can be found, such an extension of the sequence may exist (see Figure 3.6, for instance). The function of the augmentative algorithm is to examine the possibility of using the crutches to cross a barrier. The slice γ^+ in Figure 3.6 shows that not all crutches are (equally) useful. Figure 3.7 shows that more than one crutch may need to be used — in fact as many as m - 1 crutches may need to be used — to cross a barrier. The Crutch Algorithm should, therefore, be capable of examining all possible combinations of crutches that may prove useful.

Augmentation of the Modified Safeness Algorithm produces the Augmented Safeness Algorithm (ASA for brevity). This algorithm is rather complicated to follow and so it is preceded by a prologue which explains the interaction between the components of the ASA and shows a model for the working of the ASA in terms of a growing tree.

§3.7 Prologue to the Augmented Safeness Algorithm

The Augmented Safeness Algorithm is really a shell algorithm, in that it calls the Basic Algorithm iteratively until BA fails or until it is found that the terminal slice, γ_T, is accessible from the test slice.



Capacity = (25,25)

Figure 3.7

The Basic Algorithm (BA) uses the test in Theorem 3.1 to seek s slice that is accessible from the slice ω, which is one input parameter, and that possesses the prefix property with respect to ω. Occasionally, BA encounters barriers on all the chains and then it resorts to the Crutch Algorithm (CA). CA merely advances the slice to γ^+, a slice passing through a crutch, and calls BA. If BA again encounters barriers, it resorts to CA once more, and so on, so that calls to BA and CA can be nested. If BA does not encounter such barriers it seeks slices accessible from γ^+ and possessing the prefix property with respect to it. It tests these slices, γ', to determine if $\gamma' P \omega$ and if so, to declare success. If $\neg\gamma' P \omega$, it continues its search. Thus the success of BA always results in a slice, γ_p, being returned that satisfies $\gamma_p P \omega$. The slices γ' are said to be <u>conditionally acceptable</u> since it may or may not be true that $\gamma' P \omega$, but it is true that $\gamma' P \gamma^+$. If $\gamma' P \omega$, then the slice γ' is said to be <u>acceptable</u>; for instance, γ_p is always an acceptable slice.

The activity of ASA and its components can be modelled by a growing tree whose nodes represent slices. Each slice represented by a node is accessible from the slice represented by a node preceding it in the tree. The shape of a node reflects the characteristics of the slice represented. Square nodes represent acceptable slices. If a square node representing the slice γ_1 precedes a square node representing the slice γ_2, then $\gamma_2 P \gamma_1$. The test slice, σ, is at the root of the tree and is represented by a square node. An asterisk-like node represents a slice passing through a crutch relative to the slice represented by the node immediately preceding it. Triangular nodes represent conditionally acceptable nodes. If a triangular or asterisk-like node representing the slice γ_3 precedes a triangular node representing the slice γ_4, then $\gamma_4 P \gamma_3$. The plain nodes represent slices that are dead-ends.

The activity of BA appears as in Figure 3.8a, while that of the full ASA appears as in Figure 3.8b.

Readers may find Figure 3.8b of value in understanding the Augmented Safeness Algorithm.

In the statement of the ASA, the word "invocation" is used to mean "activation" and relates to recursive performances of algorithms. The term <u>hump</u> in the statement of the Crutch Algorithm refers to an arc whose demand is no less than that of the next arc.

<u>§3.8</u> <u>The Augmented Safeness Algorithm</u>

The slice whose safeness is being examined will be denoted by σ. There is an internal variable, μ, which is a slice.

<u>Step 0</u>: Set μ equal to σ. If $\mu = \sigma_T$, note that σ is safe and stop; if not, go to Step 1.

<u>Step 1</u>: Perform the Basic Algorithm with γ and ω set equal to μ and X set equal to Φ, the empty set. If the algorithm terminates unsuccessfully, go to Step 3; if not, set μ equal

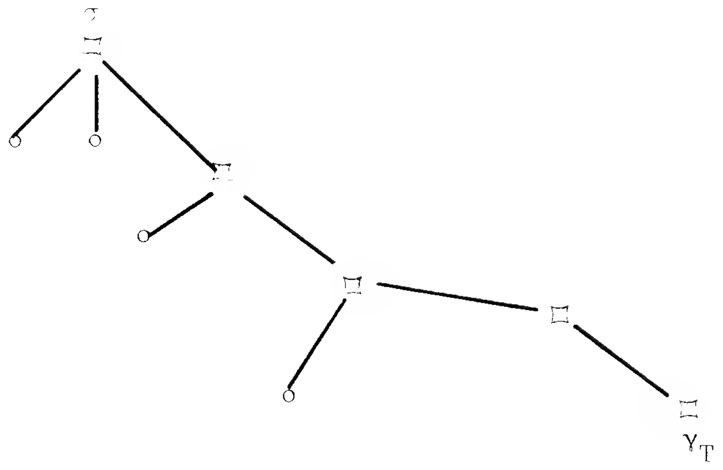


Figure 3.8a

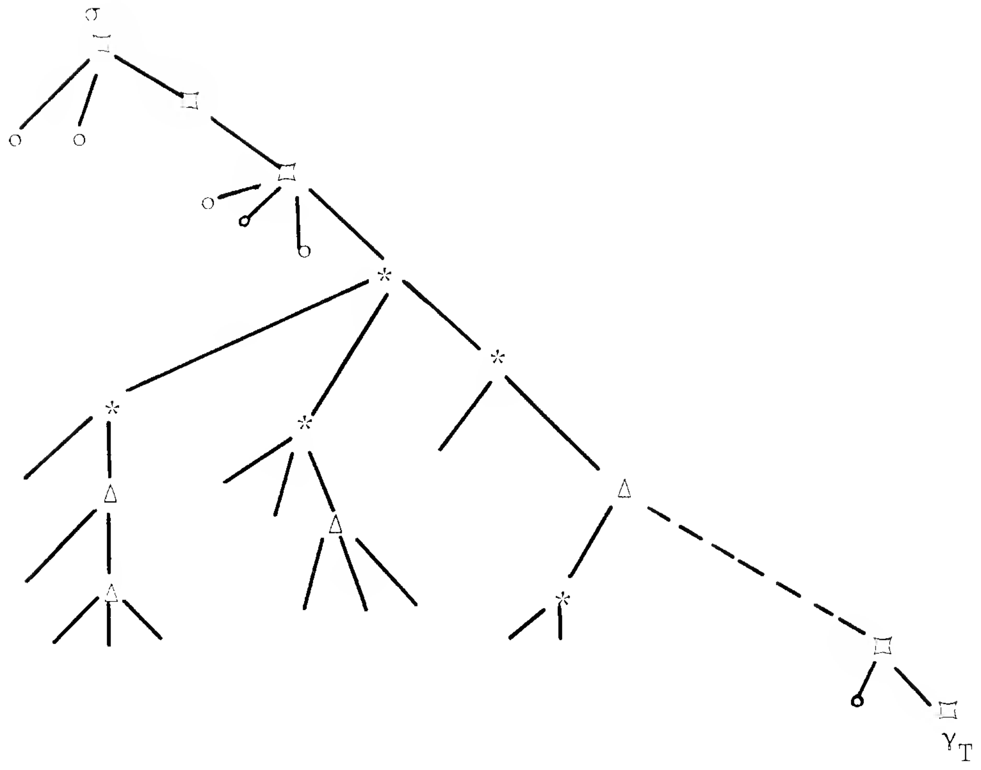


Figure 3.8b

to γ_p, the value returned, and go to Step 2.

Step 2: If $\omega = \gamma_t$, stop and report success; if not, go to Step 1.

Step 3: Report failure and stop.

Basic Algorithm

This algorithm uses three input parameters, viz two slices, γ and ω, and a set of chains, X. It seeks a feasible slice γ_p that is accessible from ω and that satisfies $\gamma_p P \omega$. (Since, presumably $\omega P \sigma$, this implies that $\gamma_p P \sigma$.) The set X_{BA} is an internal variable

Step 0: Set $X_{BA} = \{\chi_1, \chi_2, \dots \chi_m\}$. Go to Step 1 if γ is feasible; if not, terminate and report failure.

Step 1: Pick a chain from X_{BA}, preferably one that is in X — call it χ_i. Attempt to construct a uni-chain macro-move down χ_i that fits γ feasibly and is as large as possible — however, terminate the macro-move at the first point where the slice, γ', resulting from the macro-move satisfies

$$\underline{d}(\gamma' \sqcap \chi_i) \leq \underline{d}(\rho \sqcap \chi_i) \quad \text{for all slices } \rho \text{ lying between } \gamma \text{ and } \gamma' \text{ (inclusive)}$$

$$\text{and} \quad \underline{d}(S_i(\gamma') \sqcap \chi_i) \nless \underline{d}(\gamma' \sqcap \chi_i)$$

(i.e. a local minimum is reached on χ_i).

If the attempt is successful then go to Step 2. If the attempt is unsuccessful, go to Step 5.

<u>Step 2</u>: If X is empty, then go to Step 7. If X is not empty, then go to Step 3 if χ_i is not in X and to Step 4 if χ_i is in X.

<u>Step 3</u>: Set γ equal to γ' and go to Step 0.

<u>Step 4</u>: If

$$\underline{d}(\gamma' \square \chi_i) \leq \underline{d}(\xi \square \chi_i) \quad \text{for all slices } \xi \text{ lying between } \omega \text{ and } \gamma' \text{ (inclusive)}$$

then delete χ_i from X and go to Step 2. Otherwise go to Step 3.

<u>Step 5</u>: Delete χ_i from X_{BA}. If X_{BA} is now empty, go to Step 6; if not go to Step 1.

<u>Step 6</u>: Perform the Crutch Algorithm with ω, X and γ as values for the input parameters, ω^c, X^c, and γ^+, respectively. If BA terminates with success, set γ' equal to γ_c, the value returned, and go to Step 7. If BA fails, terminate and report failure.

<u>Step 7</u>: Set γ_p equal to γ', terminate and report success.

<u>Crutch Algorithm</u>

This algorithm extends the sequence of slices to a slice which passes through a crutch relative to the input parameter γ^+. It uses an internal variable X_{CA} which is initialised to $\{\chi_1, \chi_2, \dots \chi_m\}$ at entry. It uses the input parameters ω^c and X^c for calls to BA.

<u>Step 0</u>: Pick a chain from X_{CA} — call it χ_j. Go to Step 1.

<u>Step</u> <u>1</u>: Attempt to construct a uni-chain macro-move down χ_j that fits γ^+ feasibly and that is as large as possible — however, terminate the macro-move at the first point where the slice, γ^*, produced by the macro-move satisfies:

$$\text{(i)}\quad \underline{d}(\gamma^* \mathbin{\square} \chi_j) \not> \underline{d}(\gamma^+ \mathbin{\square} \chi_j) \ \underline{\text{or}}\ \exists\alpha_j \ \ \underline{d}(\alpha_j) > \underline{d}(\gamma^* \mathbin{\square} \chi_j) \text{ where } \gamma^+ \mathbin{\square} \chi_j < \alpha_j < \gamma^* \mathbin{\square} \chi_j$$

i.e., either γ^* contains a crutch or a hump α_j was crossed.

and (ii) $\underline{d}(\gamma^* \mathbin{\square} \chi_j) \not> \underline{d}(S_j\ (\gamma^*) \mathbin{\square} \chi_j)$

If the attempt succeeds, go to Step 2; if not, go to Step 3.

<u>Step</u> <u>2</u>: Add χ_j to X^c and call for the performance of BA with the input parameters γ^*, X^c and ω^c as values for the input parameters γ, X and ω. If BA terminates successfully, then set γ_c equal to the value, γ_p, returned by BA and go to Step 5. If BA terminates unsuccessfully (then the macro-move $\gamma^+ \to \gamma^*$ is not acceptable and so), set γ^+ equal to γ^* and go to Step 1 (rather than to Step 0 as a larger uni-chain macro-move down χ_j than $\gamma^+ \to \gamma^*$ may be acceptable).

<u>Step</u> <u>3</u>: Delete χ_j from X_{CA}. If X_{CA} is now empty, go to Step 4; if not, go to Step 0.

<u>Step</u> <u>4</u>: Terminate and report failure.

<u>Step</u> <u>5</u>: Terminate and report success.

§3.9 Adequacy of ASA

Theorem 3.2, which follows, shows that the Augmented Safeness Algorithm is sufficiently potent to handle all vector demand graphs.

THEOREM 3.2 The Augmented Safeness Algorithm applied to a slice of a vector demand graph terminates successfully if and only if the slice is safe.

PROOF: The "only if" result follows from the fact that ASA terminates successfully only if the terminal slice of the graph is reached and from the fact that every slice in the sequence constructed is feasible.

It remains to be shown that ASA never reports failure erroneously, i.e. when the slice being tested is safe.

Suppose ASA failed even though the slice under test is safe.

Let D be the demand graph and σ the slice under test. Now failure of ASA implies failure of BA, which implies failure of CA. Let γ_t represent the last value of γ_p returned by BA. Then γ_t is the last slice possessing the prefix property with respect to σ that was found by ASA. At γ_t, all attempts by BA to use the test of Theorem 3.1 failed and BA asked for the performance of CA, which reported failure. That CA reported failure when applied at γ_t implies that all attempts to use crutches failed sooner or later.

In terms of the tree of Figure 3.8b, the (sub-) tree rooted at γ_t contains asterisked, triangular and plain nodes only. The leaves of the tree are plain nodes and these slices have the property that there are no accessible crutches below them, i.e. there are m barriers β_i^δ (on the m chains) relative to each such slice δ. The arcs between $\delta \sqcap \chi_i$ and β_i^δ all have demands strictly greater than that on $\delta \sqcap \chi_i$. Let β_i, for all m values of i, be the lowest of the barriers β_i^δ, i.e.

$$\beta_i \succcurlyeq \beta_i^\delta \quad \text{for all slices of the form } \delta$$

Since σ is safe and since γ_t possesses the prefix property with respect to σ, γ_t is safe. Thus there exists a connected sequence, Σ, of feasible slices from γ_t to γ_T. Let σ' be the first slice in Σ to pass through one of the β_i's. Say σ' passes through β_k. Then

$$\gamma_t \sqcap \chi_j \preccurlyeq \sigma' \sqcap \chi_j \prec \beta_j \qquad j \in [1, m] \qquad j \neq k$$

$$\gamma_t \sqcap \chi_k \prec \sigma' \sqcap \chi_k = \beta_k \qquad k \in [1, m]$$

It will now be shown that the connected sequence of feasible slices $\gamma_t \ldots \sigma'$ can be transformed into one that can be produced by ASA. Since ASA was unable to produce it, a contradiction will result. This will imply that a sequence such as Σ cannot exist.

Let the macro-move $\gamma_t \rightarrow \sigma'$ be broken up into uni-chain macro-moves, μ_1, μ_2, ... μ_q, so that

$$\gamma_t \; \mu_1\mu_2 \; \cdots \; \mu_q = \sigma'$$

Consider an intermediate slice, γ', in the sequence $\gamma_t \to \sigma'$ that terminates a uni-chain macro-move, and say

$$\gamma' = \gamma_t \; \mu_1\mu_2 \; \cdots \; \mu_\ell$$

Let γ'' represent the slice $\gamma_t \; \mu_1\mu_2 \; \cdots \; \mu_{f+1}$ and say μ_{f+1} is a macro-move on χ_ℓ.

Then two cases can arise:

<u>Case 1</u> $\quad d(\gamma'' \sqcap \chi_\ell) \not> \underline{d}(\gamma' \sqcap \chi_\ell)$

In this case μ_{f+1} consists in moving to a relative crutch, i.e. a crutch relative to γ'. (It should be noted that an arc α_ℓ which satisfies:

$$\underline{d}(\alpha_\ell) \leq \underline{d}(\gamma' \sqcap \chi_\ell)$$

is also a crutch with respect to γ.) In this case, μ_{f+1} is to be left unchanged.

<u>Case 2</u> $\quad \underline{d}(\gamma'' \sqcap \chi_\ell) > \underline{d}(\gamma' \sqcap \chi_\ell)$

In this case $\gamma'' \sqcap \chi_\ell$ is not a relative crutch. Here two sub-cases arise:

<u>Case A</u> There is an arc α_ℓ on χ_ℓ, between γ' and γ'', that satisfies

$$\underline{d}(\alpha_\ell) \not\leq \underline{d}(\gamma'' \sqcap \chi_\ell)$$

i.e. a hump was crossed.

In this case μ_{f+1} is left unchanged.

<u>Case B</u> There is no arc on χ_ℓ, between γ' and γ'', that satisfies

$$\underline{d}(\alpha_\ell) \nless \underline{d}(\gamma'' \square \chi_\ell)$$

In this case the arc $\gamma'' \square \chi_\ell$ has the greatest demand of all arcs on χ_ℓ between γ' and γ'' (inclusive). Two further cases can arise.

<u>Case B1</u> There is an arc α'_ℓ on χ_ℓ, between γ' and γ'' and as close to γ'' as possible, that satisfies:

$$\underline{d}(\alpha'_\ell) \ngtr \underline{d}(\gamma' \square \chi_\ell)$$

and $\underline{d}(S_\ell(\alpha'_\ell) \nless \underline{d}(\alpha'_\ell)$

In this case, μ_{f+1} is shortened to stop at a slice passing through α'_ℓ. Let the remaining part of μ_{f+1} be labelled μ'_{f+1}.

<u>Case B2</u> There is no such arc α'_ℓ.

In this case μ_{f+1} is shortened to λ, the null move. Let the remaining part (i.e. μ_{f+1}) be labelled μ'_{f+1}.

In either of the two cases B1 and B2 above, no point is served in carrying out μ'_{f+1} immediately after μ_{f+1}, and μ'_{f+1} can be consolidated with any later uni-chain macro-move, μ_h, down χ_ℓ. For the macro-move $\mu_{f+2} \cdots \mu_h$ still fits $\gamma_t\mu_1\mu_2 \cdots \mu_{f+1}$ feasibly, as the demand on $(\alpha'_\ell/\gamma'' \square \chi_\ell)\gamma''$

is no greater than that on γ''. (If there is no later move down χ_ℓ and $\ell \neq k$, then μ'_{f+1} can be dropped, while if $\ell = k$ then f'_{f+1} can be carried out towards the end of the sequence $\gamma_t \to \sigma'$, i.e. after μ_q.)

To summarise Cases 1 and 2, either μ_{f+1} consists in moving to a relative crutch or in crossing a hump, or μ_{f+1} can be shortened to consist in moving to a relative crutch. (The shortening may reduce μ_{f+1} to a null move λ.) In any case, μ_{f+1} is or can be made a move of the kind that the Augmented Safeness Algorithm produces.

Let the uni-chain macro-moves when consolidated and transformed be labelled by μ's with asterisks, so that μ_{f+1}, for instance, becomes $\overset{*}{\mu}_{f+1}$. Then it is clear from the discussion above that the slice resulting from the application of any macro-move, $\overset{*}{\mu}_1\overset{*}{\mu}_2 \ldots \overset{*}{\mu}_f$, to γ_t has a demand no greater than the demand of a slice resulting from the application of the corresponding macro-move $\mu_1\mu_2 \ldots \mu_f$, to γ_t. Thus the macro-move $\mu_{f+1}\mu_{f+2} \ldots \mu_q$ (ignoring moves already made) fits $\gamma_t\overset{*}{\mu}_1\overset{*}{\mu}_2 \ldots \overset{*}{\mu}_f$ feasibly and, therefore, so too does $\overset{*}{\mu}_{f+1}\overset{*}{\mu}_{f+2} \ldots \overset{*}{\mu}_q$ followed by $\mu'_1\mu'_2 \ldots \mu'_q$ (ignoring those included in a $\overset{*}{\mu}$ due to consolidation).

One thus gets a sequence of uni-chain moves, of the kind ASA that generates, which leads from γ_t to σ' by means of feasible slices alone.

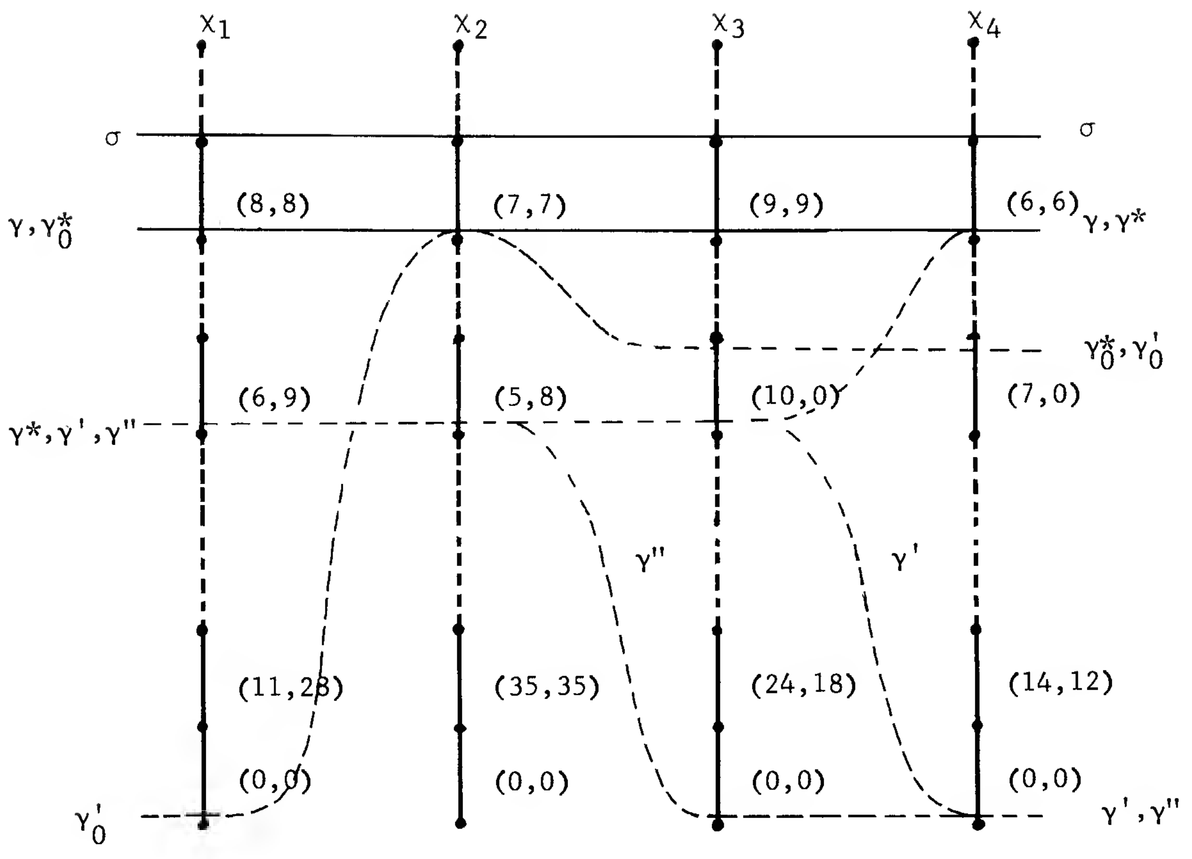
But this is absurd, since β_k is the lowest of the barriers discovered on χ_k by ASA!

Thus the sequence Σ cannot exist and γ_t must be unsafe. As γ_t possesses the prefix property with respect to σ, this implies that σ is unsafe. Thus, the "if" part of the result in the theorem has been proved.

Q.E.D.

The reader who is still skeptical about the necessity for the complicated interactions and backtracking in the Augmented Safeness Algorithm, should remember that the algorithm is expected to handle all cases and, in particular, the case illustrated in Figure 3.9. In that figure it will be seen that if a choice of crutches is made so that one reaches γ^*, then two conditionally acceptable slices γ' and γ'' can be found (which possess the prefix property with respect to γ^*) before it is realized that there is no way in which γ_T can be reached from γ''. It is necessary then, to backtrack to γ and (perhaps with some further fumbling) move to γ_o^* instead of γ^*. The sequence of slices $\gamma \ldots \gamma_o^* \ldots \gamma_o'$ illustrates that γ_T can be reached from γ by way of γ_o^*.

Careful observation of the Augmented Safeness Algorithm, and the Crutch Algorithm in particular, shows that in the worst case it tries out all possible crutch combinations in an enumerative manner. It is interesting that this is not a fault of the way the algorithm works. This is stated more precisely in Theorem 3.3 below. A few definitions and a lemma lay the groundwork for Theorem 3.3 and these follow.



Capacity = (35,35)

<u>Figure 3.9</u>

§3.10 Characterization of Safeness Algorithms

By algorithm is meant an algorithmic test for the safeness of an arbitrary slice of an arbitrary demand graph that attempts to construct a connected sequence of feasible slices from the test slice to the terminal slice of the demand graph.

A local algorithm is one which, at any point in the construction of a connected sequence, has the partial sequence of slices constructed up to that point as the only information about the demand graph on which to base its decision regarding what move to try next. Thus, a local algorithm does not know about the entire remaining portion of the demand graph and, therefore, cannot make only the correct move (in the defined technical sense) every time. Similarly, a local algorithm does not have recall abilities in respect of futile past moves other than to recall that they were futile. Thus, it cannot sweep down the chains one at a time and thereby gain (and store) knowledge of the whole or part of the remaining portion of the demand graph. (Were one to assume such an ability, then it is clear that an arbitrarily large memory would be required to store the information, as the chains can be of arbitrary length. Since any realistic memory has finite capacity, such an assumption is clearly unrealistic.) It can be seen, easily, that both the Modified and Augmented Safeness Algorithm are local. If the order $\chi_1, \chi_2, \dots \chi_m$ is used whenever chains are to be picked, then this obviates the need for recording futile use of chains. The use of the set

χ of preferred chains is not crucial to the working of ASA — it merely makes ASA more efficient.

A local algorithm is said to be a limited-backtracking algorithm, if one can generally partition the sequence of slices it produces into two or more sub-sequences, the initial and terminal slices of which possess the prefix property with respect to the slice whose safeness is being investigated. The Safeness Algorithm of Chapter 2 and the Augmented Safeness Algorithm are limited backtracking algorithms. An equivalent definition of a limited backtracking algorithm is one that states that the sequence of moves constructed can be broken up into macro-moves such that each such macro-move is applied to and produces a slice possessing the desired prefix property. Let these macro-moves be called correct macro-moves. A limited backtracking algorithm is said to be linear if the number of macro-moves examined, before the correct macro-move to apply at an intermediate point, characterized by the slice γ, is found (or it is discovered that none exists), is always less than

$$A.f(n_1, n_2, n_3, \ldots n_m)$$

where: A is some constant, $f(n_1, n_2, \ldots n_m)$ is linear in the n_i, i.e. of the form $\sum_{i=1}^{m} a_i n_i$ (where the a_i are integer constants), and n_i is the number of relevant arcs on chain χ_i below γ. If the function f increases with the n_i faster than any linear bound does, then the algorithm is said to be of higher order or non-linear.

In the case of the Safeness Algorithm of Chapter 2, the number of macro-moves examined at a time is at most m, i.e. $A = m$ and $f(n_1, n_2, \ldots n_m) = 1$, and the algorithm is thus linear. An example of an algorithm of higher order is the Augmented Safeness Algorithm. (This statement is clarified in the theorem below.) In the case of the Augmented Safeness Algorithm, the relevant arcs are crutches with respect to γ, so that n_i is the number of such crutches on χ_i.

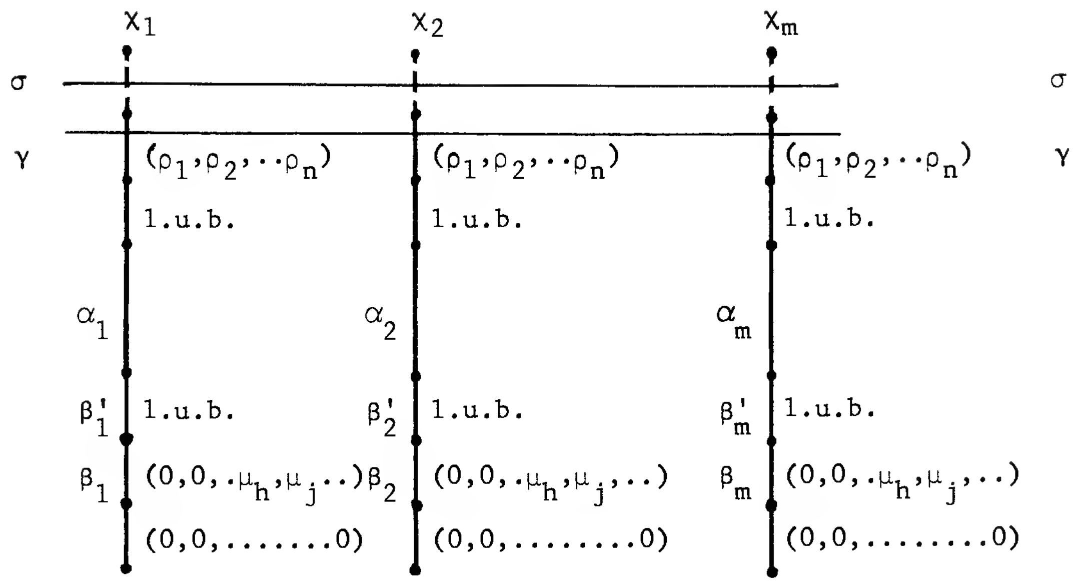
The lemma which follows is essential to the proof of Theorem 3.3.

<u>LEMMA 3.4</u> Let D be the demand graph defined by Figure 3.10. The arcs marked $\beta_1', \beta_2', \ldots \beta_m'$ are m barriers on the m chains. The arcs marked l.u.b. are arcs whose demands are the least upper bounds of the demands on the two arcs on either side of these arcs. The arcs marked α_i are crutches.

Let γ_p' be a feasible slice that is accessible from γ and is distinct from γ. If γ_p' lies above the barrier slice $\beta_1'\beta_2' \ldots \beta_m'$, then γ_p' cannot possess the prefix property with respect to γ.

<u>PROOF</u>: Since γ_p' is accessible from γ, the macro-move $\gamma \rightarrow \gamma_p'$ fits γ feasibly. Let this macro-move be broken up into uni-chain macro-moves so that $\gamma \rightarrow \gamma_p' = \mu_1\mu_2 \ldots \mu_q$.

Let μ_q be a macro-move down chain χ_ℓ, and let $\gamma\mu_1\mu_2 \ldots \mu_{q-1}$ be referred to as γ_{q-1}. Then there are two cases:



Capacity = $(C_1,C_2,....C_n)$

$\mu_j = [\ C_j - (m-1)\rho_j\ ] + k$; $\mu_j > 0$; $C_j \geq k.\rho_j + (m-k).(\rho_j+1)$

$\mu_h = [\ C_h - (m-1)\rho_h\ ] - k$; $\mu_h \geq 0$; $C_h \geq (m-k).\rho_h + k.(\rho_h+1)$

Of the arcs α_i:

(i) Exactly k have the demand $(\rho_1,\rho_2,..\rho_h+1,..\rho_j-1,..\rho_n)$

(ii) The rest have the demand $(\rho_1,\rho_2,..\rho_h-1,..\rho_j+1,..\rho_n)$

The critical resource is the j^{th}; the h^{th} component is specified so as to ensure that each α_i is a crutch relative to γ but $\underline{d}(\alpha_i) \not\leq \underline{d}(\gamma \sqcap \chi_i)$.

Figure 3.10

<u>Case 1</u> $\underline{d}(\gamma'_p \square \chi_\ell) \not\leq \underline{d}(\gamma_{q-1} \square \chi_\ell)$

In this case γ_{q-1} meets all the conditions of Lemma 3.1 and, therefore, γ'_p cannot possess the prefix property with respect to γ.

<u>Case 2</u> $\underline{d}(\gamma'_p \square \chi_\ell) \leq \underline{d}(\gamma_{q-1} \square \chi_\ell)$

In this case $\gamma_{q-1} \square \chi_\ell$ must be one of the arcs marked "l.u.b.". Thus $\gamma_{q-1} \square \chi_\ell$ must have a greater demand than the arc, α', preceding it, viz an α or the arc $\gamma \square \chi_\ell$.

Let μ_f be the previous move down χ_ℓ. Then $\gamma\mu_1 \cdots \mu_{f-1} \square \chi_\ell$ must lie above α' or be α'.

If $\gamma\mu_1 \cdots \mu_{f-1} \square \chi_\ell$ is α' then the move μ_f can be deleted at this point. Since $\mu_{f+1} \cdots \mu_{q-1}$ will fit $\gamma\mu_1 \cdots \mu_{f-1}$ feasibly. Let $\gamma\mu_1\mu_2 \cdots \mu_{f-1}\mu_{f+1} \cdots \mu_{q-1}$ be γ_o. Then γ'_p is accessible from γ_o since the move μ'_q, which is μ_f consolidated with μ_q, fits γ_o and leads to γ'_p. Thus γ_o meets all the conditions of Lemma 3.1 and, therefore, γ'_p cannot possess the prefix property with respect to γ.
If $\gamma\mu_1\mu_2 \cdots \mu_{f-1} \square \chi_\ell$ lies above α', then μ_f can be shortened so that $\gamma\mu_1\mu_2 \cdots \mu_f$ includes α'. Once again $\gamma\mu_1\mu_2 \cdots \mu_{q-1}$ meets all the conditions of Lemma 3.1 and, therefore, γ'_p cannot possess the prefix property.

Q.E.D.

The values of demand have been chosen so that there is exactly one set of k crutches which must be used to cross any barrier at all. Non-obstructive arcs are arcs whose demand vectors are such that any feasible slice that includes ℓ arcs that are crutches, for any $\ell \leq m - 1$, (and shares the remaining arcs with γ) is accessible from γ. The arcs marked l.u.b. are non-obstructive arcs. For if γ_o is a slice going through ℓ crutches, then $(\gamma \square \chi_i/\alpha_i)\gamma_o$, where α_i is a crutch, is feasible and so is γ_o. That any feasible slice that uses ℓ crutches in addition to arcs from γ is accessible from γ will be used below.

Since a local algorithm has no way of knowing which combination of crutches is correct other than by trial and error, as many as $Z - 1$ trials can be wasted, where Z is the number of possible crutch combinations of from 1 to $m - 1$ crutches (one from each chain) at a time that correspond to slices accessible from γ. Here $n_i = 1$, for all values of i, and since all slices using ℓ crutches are accessible, $Z = 2^{m-1}$.

The non-linearity of a local limited-backtracking algorithm is thus obvious.

Q.E.D.

To further simplify understanding of the example, Figure 3.11 shows a special case of Figure 3.10.

The construction of Figure 3.10 is quite general, in that k can be an arbitrary integer between 1 and m-1 and can be chosen suitably. Now suppose a limited back-tracking algorithm is given. Since it is local, it must examine the combinations of the crutches in some order, and for each combination of r crutches it tries out some moves. However, since there is only one combination that works, all other trials are wasted. The number of trials wasted can be made non-linear by choosing a value of k appropriate to the algorithm. (It should be noted that the choice of values for C_j and C_h ensures that all slices which use from 1 to m-1 crutches are feasible and accessible from γ.)

For example, consider an algorithm that uses the crutches 1 at a time, 3 at a time, etc. up to m-1 or m-2 (whichever is odd) at a time and then 2, 4, 6 ... at a time.

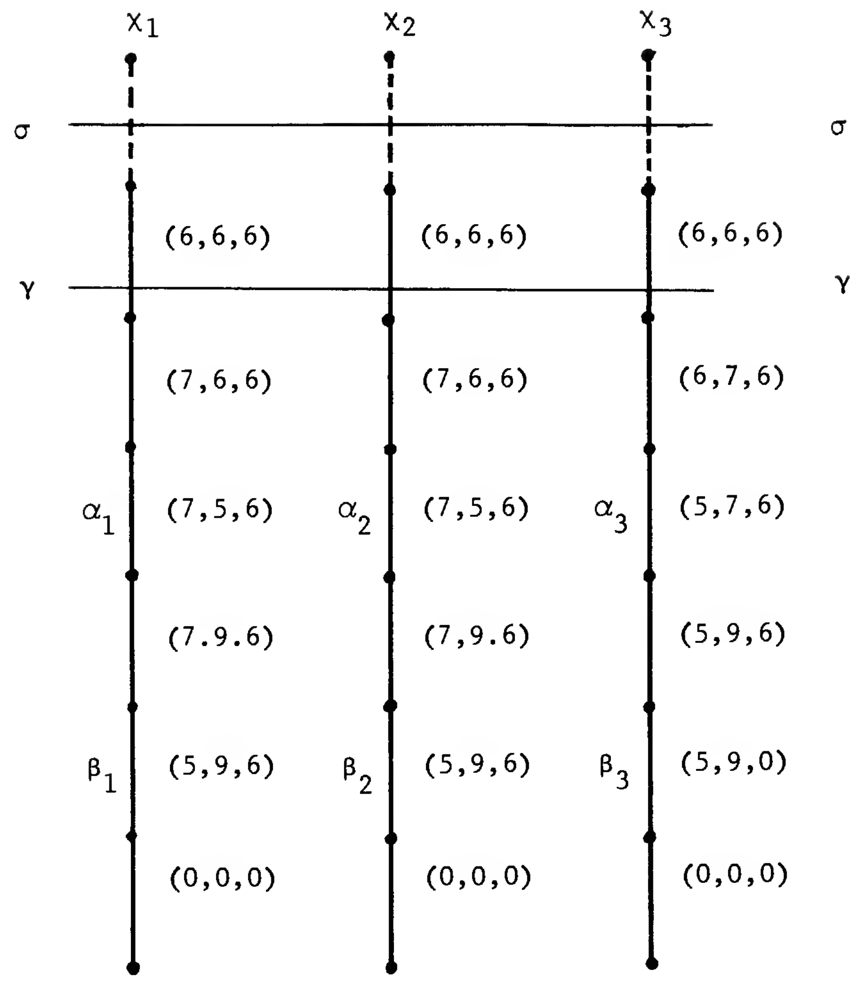
Pick k = 2. Then the number of wasted trials

$$= \sum_{\substack{1 \\ (r \text{ odd})}}^{m'} (\text{no. of combinations} - r \text{ crutches at a time}) \text{ where } m' = \begin{matrix} m-1 \text{ or} \\ m-2 \end{matrix}$$

$= $ the sum of the coefficients of $x^1, x^3, x^5, x^7 \ldots x^{n-1}$

in $(1 + x)\ (1 + x)\ (\quad)\ (\quad) \ldots (1 + x)$

The right hand side is 2^{m-1}, which is non-linear in m.



Capacity = (19,19,19)

(With reference to Figure 3.10,m=3,n=3,k=2,h=1,j=2)

The 'correct' combination of crutches is α_1 and α_2.

<u>Figure 3.11</u>

Comment 1: The proof technique above is really quite conservative for Figure 3.9 shows that merely being able to cross the barrier is not a guarantee of being able to reach a slice that possesses the prefix property (without further backtracking).

Comment 2: It is clear that if no combination of crutches (from 1 to m-1 of them) permits crossing of any barrier, then γ (and hence σ) is unsafe.

The theorem above indicates that the Augmented Safeness Algorithm is in a sense optimal. As long as the Basic Algorithm succeeds the number of sequences examined in vain is at most m-1 and consequently the algorithm is linear. When it fails, it is necessary for the Crutch Algorithm to try crutches in a trial and error fashion to get past the barriers discovered earlier by the Basic Algorithm. It then tries to reach a slice possessing the prefix property (by use of the Basic Algorithm); the Basic Algorithm can then be used again.

The rest of this chapter deals with special cases of the rectilinear vector demand graphs discussed so far.

§3.11 Locked Data Bases and Semaphores

One of the resources that can be shared in an unpreemptible manner in computer systems is a set of data bases that have locks on them; only one user or process at a time can use such a data base. Tables of miscellaneous varieties in the operating system software are typical entities of this kind.

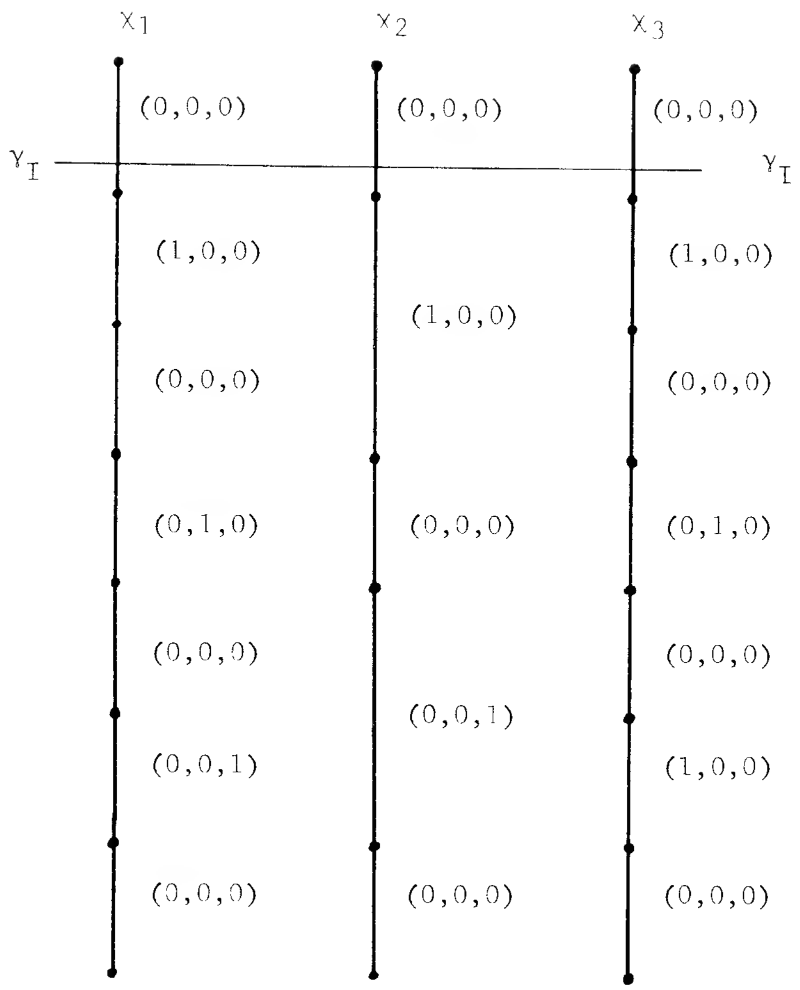
The lock is exactly analogous to Dijkstra's semaphores [10]. A process examines the lock to see if it is set; if it is not set (the corresponding semaphore has value 1) then it is set (the semaphore is decremented by 1). The lock stays set until the process using the data base relinquishes control — at this time the lock is reset again. Of course, semaphores are more general than locks, in that they can be used for coordination of activities in general. However, whether processes use semaphores or locked data bases, deadlocks can occur. The corresponding demand graphs have demand components which are always either 0 or 1 and C is (1, 1, ... 1). The techniques described in this chapter can be used to examine the consistency of use of semaphores (or locked data bases) by a set of users or processes in such a system.

§3.12 Job Shop Scheduling

A problem of considerable interest in the field of operations research is that of scheduling a set of manufacturing jobs in a workshop. Say the workshop processes raw stock of some kind in several steps to produce useful items. There could be variations in processing for different raw stocks and different items. In any case, one can draw up a job chart, which describes which processes have to be performed and in what order. The jobs are then to be scheduled on the different machines that do the processing.

One can represent the jobs by a demand graph of the kind shown in Figure 3.12. Each arc has a demand consisting of zero's and one's corresponding to the machines it does not and does need, respectively, in that phase. The General n/m Job-Shop Problem [11] deals with n jobs and m distinct machines — in this case the demand graph has n chains and each demand vector has m components (the interchanged notation is confusing and regrettable). $\underline{C}$ is (1, 1, 1, ... 1), indicating that there are m distinct machines. Thus the Job Shop can be represented by a restricted class of demand graphs.

However, it is important to note that each arc of the demand graph of Figure 3.12 that has a non-zero associated demand is followed by an arc with a zero associated demand. This is true for all Job Shop problems, as the operations are performed one at a time and jobs can lie between two machines — having been processed by one (freeing that machine for other work) and awaiting processing by the other. But this feature automatically ensures that any slice of the demand graph that is feasible is also safe! Thus deadlocks and examination of safeness are not important issues in Job Shops. Rather, it is the minimization of processing time (average or maximum) for a set of jobs that is an interesting problem — particularly, as the time required for each operation is quite predictable.



Capacity = (2,1,1)

Figure 3.12

Demand Graphs For Systems With Interacting and Internally Parallel Activities

Chapter 4

§4.1 Arboraceous Demand Graphs

In this chapter the constraints on components of demand graphs are relaxed somewhat. The components will look like trees but with more than arc incident on some nodes. Since the word tree[†] has been used to describe what others[#] call arborescences, both terms will be avoided. Instead the term arbour will be used. An arbour is a finite directed graph that is circuit free, i.e., that has no directed cycles. An arbour always has at least one node with indegree zero and one with outdegree zero. An arboraceous demand graph is a demand graph whose components are arbours and whose arcs are labelled with demands chosen from the set of n-tuples of integers. The capacity associated with the graph is also such an n-tuple. No distinction will be made between vector and scalar demands on arboraceous demand graphs, except where exceptional properties appear in graphs with scalar demands. Initial and terminal arcs are respectively out-going arcs of transitions with zero indegree and in-coming arcs of transitions with zero outdegree. Initial and terminal arcs have zero demand. Transistions with indegree one and outdegree greater than one are called forks after Conway [14]. Transitions with indegree greater than one are known as points of synchronisation or points of interaction. Every point of synchronisation must have at least one outgoing arc.

[†]See [12] for instance

[#]See [13] for instance

In terms of systems of processes sharing resources, arboraceous demand graphs represent systems in which processes are not necessarily either sequential or independent. Such systems, with parallel or interacting processes or processes that are both, are not uncommon. In terms of the construction analogue of Chapter 1, contractors may undertake more than one project at a time, with the projects sharing initial or final phases of activity but being independent otherwise. Alternatively, some projects may be too large for one contractor and may be undertaken by several contractors with division of the work into independent sequences of tasks with some interaction between contractors. In computer systems such as MULTICS [15] processes can produce other processes and interact with each other by means of the "block" and "wake-up" primitives. The interaction that has been mentioned so far is explicit interaction, that is interaction other than through the sharing of limited resources. There is one kind of interaction, however, that is modelled like explicit interaction even though it is occasioned by resource sharing. This is mechanism for acquisition of write access capability in systems which guarantee determinacy of computations — such as those of Van Horn [16], the implementation in MULTICS of which is discussed by the author in [17]. In Van Horn's systems, a clerk (process) which possesses read-access capability for a shared data object acquires write access capability for it when every other clerk has relinquished its read access capability. This behaviour cannot be modelled merely by treating such a data object as one kind of resource.

Rather, the dependency of the first process on the others has to be modelled explicitly, as in Figure 4.1 where the process represented by the chain that begins with α_2 is the one that waits to acquire write access capability before proceeding with the phase represented by α_2'.

When arboraceous demand graphs represent systems of users, the users are not in one to one correspondence with the components of the demand graph; for two or more interacting users appear as one component. Rather, the only construct in the demand graph that indicates the number of users in the system represented is the number of initial arcs. If every user's processes merge or join [14] before his activity terminates, then the number of terminal arcs in the demand graph representing the system also indicates how many users the system has.

§4.2 Slices and Related Concepts

A sliver in an arboraceous demand graph is a cut-set of a component of the demand graph. A slice of an arboraceous demand graph is a set of slivers, one from each component-graph. Slices are denoted by lower case Greek letters other than α and β — usually γ. The pendant sub-graph of an arc consists of the arc and the arbour from its terminal transition, t, i.e., the maximal arbour, with t as the only transition with zero indegree, that is a sub-graph of the graph. The

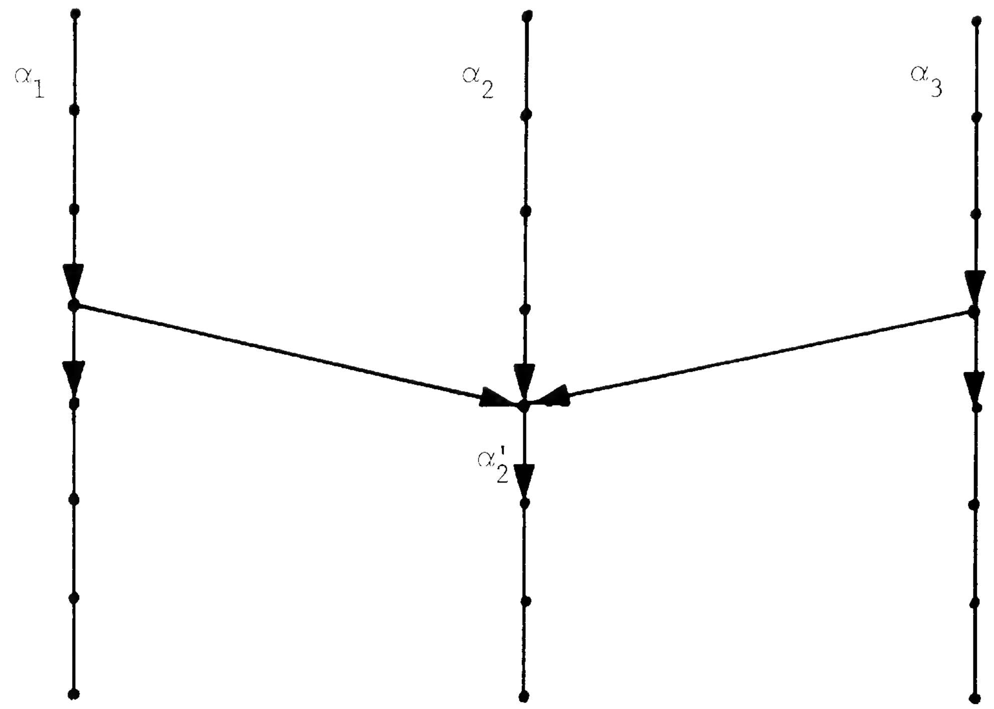


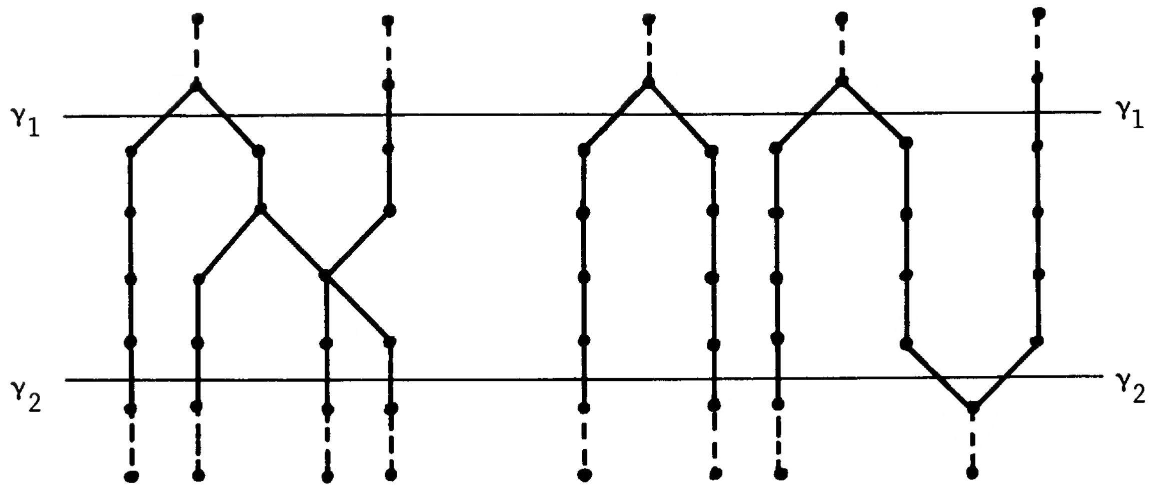
Figure 4.1

pendant sub-graphs of the arcs in a slice of an arboraceous demand graph are termed the chain-graphs defined by the slice. Clearly, the chain-graphs defined by a slice are not necessarily disjoint. In rectilinear demand graphs chain-graphs are chains and this is what suggests the terminology for arboraceous demand graphs. Chain graphs are represented by χ.

As in Chapter 2, since a slice of an arboraceous demand graph partitions the transitions of the graph, one can speak of the predecessor set and successor set of a slice. The relations "earlier than or the same as" and "later than or the same as" for slices are represented by "$\leq$" and "$\geq$", respectively, and are defined exactly as in Chapter 2.

The initial slice, γ_I, and terminal slice, γ_T, of a demand graph are defined as in Chapter 2.

A frustum of a demand graph is the part of the graph that lies between two slices, one of which is earlier than the other. The frustum defined by slices γ_1 and γ_2 of a demand graph D, where $\gamma_1 \preccurlyeq \gamma_2$, is denoted by $F(D, \gamma_1, \gamma_2)$. A frustulum is a component of a frustum. The frustula of $F(D, \gamma_1, \gamma_2)$ are denoted by $f_j(D, \gamma_1, \gamma_2)$, for the j^{th} frustulum, or simply f_j when the frustum referred to is clear from the context. By analogy to entire demand graphs, cut-sets of frustula are also termed slivers — the components of the demand graph are the frustula of $F(D, \gamma_I, \gamma_T)$. In rectilinear demand graphs, frustula are chains. Figure 4.2a shows a frustum of a demand graph and Figure 4.2b shows the frustula of the frustum. As indicated in Figure 4.2



(The dashed arcs are not in the frustum)

Figure 4.2a

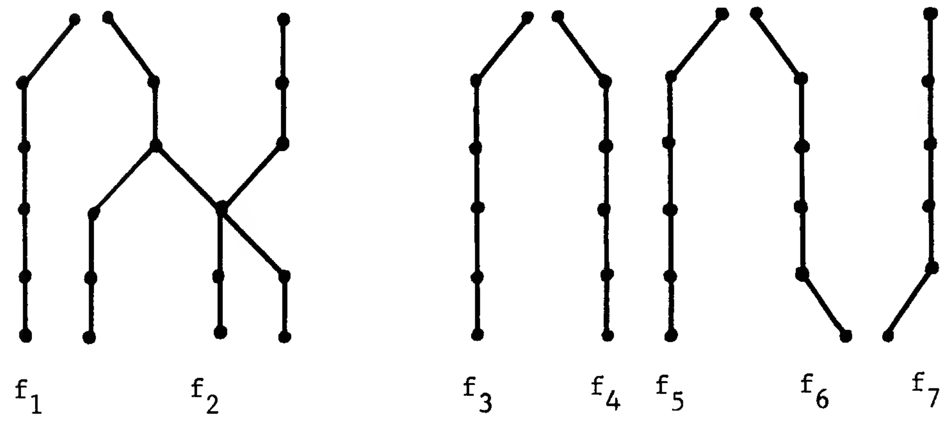


Figure 4.2b

transitions immediately following the slice γ_2 and immediately preceding the slice γ_1 are part of the frustum $F(D, \gamma_1, \gamma_2)$, but forks preceding γ_1 are split up into as many nodes as there are outgoing arcs and points of interaction are split up into as many nodes as there are incoming arcs. As a consequence, in Figure 4.2b, the subgraphs marked f_3 and f_4 or those marked f_6 and f_7 are distinct frustula.

The concepts of immediate-successor slices, moves, macro-moves, uni-chain macro-moves, connected sequences of slices, runs, feasibility and safeness of slices, etc., carry over directly from Sections 2.6 and 2.9.

The slices of an arboraceous demand graph representing a system correspond, as before, to the states of the system. The number of chain-graphs defined by the current slice corresponds to the number of processes in the system in the current state. As before, a state is also called an allocation state and feasible slices represent meaningful allocation states. Safe slices represent states from which the processes can be scheduled so as to run to completion without deadlock. In general, the interpretations of Chapter 2 carry over. However, the term "user" is now not necessarily synonymous with the term "process" since a user's activity may involve several processes, even though it involves only a single process initially.

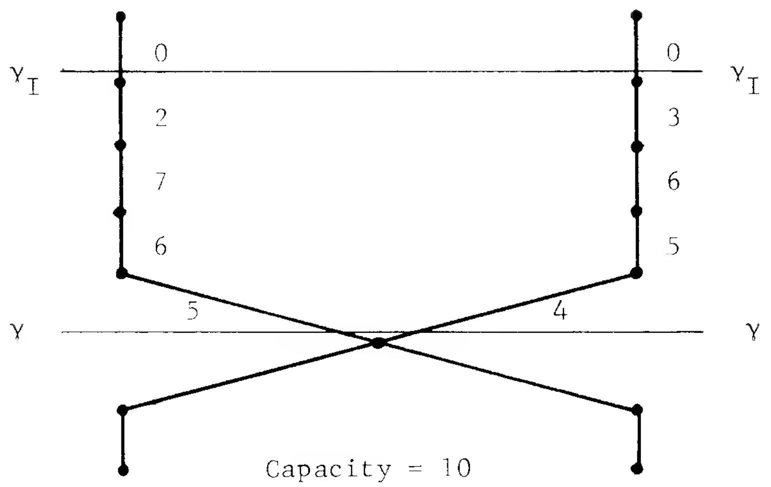


Figure 4.3a

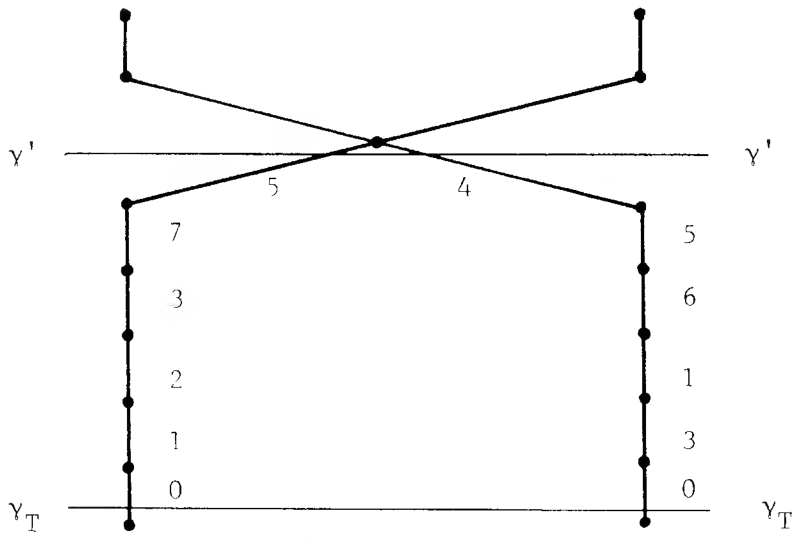
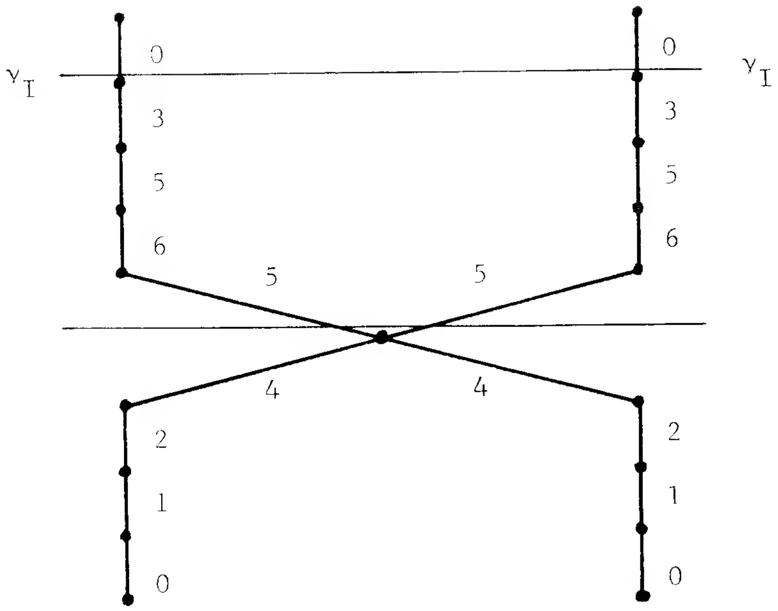
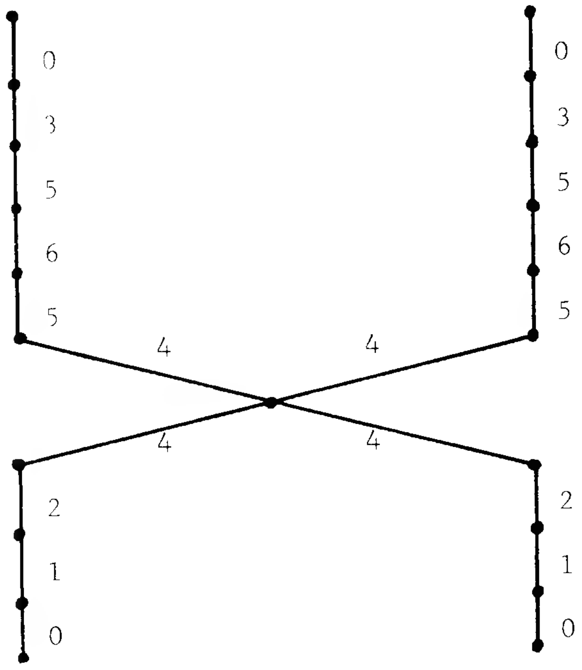


Figure 4.3b



Capacity = 10

Figure 4.4a



Capacity = 10

Figure 4.4b

then the graph of Figure 4.4a can be transformed into that of Figure 4.4b which does not exhibit inherent deadlock. There are instances, however, when such a transformation does violence to the representation. For the processes may deliberately be withholding resources from other processes until certain conditions are satisfied by the latter processes, satisfaction of the conditions being signalled by the processes reaching the point of interaction. Consequently, although it is tempting to prescribe a transformation of arboraceous demand graphs so as to duplicate the arcs preceding and following a point of synchronisation and replace the demand on the one near the point by the g.l.b. of the demands on the arcs on either side of the point, no transformation will be prescribed. However, the spirit of the transformation should be borne in mind in the specification of a demand graph for a system of processes.

§4.5 The Prefix Property

As with rectilinear demand graphs, it is desirable to have limited-backtracking algorithms for determination of the safeness or unsafeness of a slice. This requires extension of the prefix property to arboraceous demand graphs.

The set of extensions $E_D(\gamma)$ of an arboraceous demand graph is the set of arboraceous demand graphs that are identical to D until γ, and that have the same capacity as D. An element of $E_D(\gamma)$ is an extension of D with respect to γ. If D' is such an extension, then

$F(D', \gamma_I', \gamma) = F(D, \gamma_I, \gamma)$, where γ_I' is the initial slice of D'.

The definition of the prefix property is identical to that in Chapter 3.

§4.6 Necessary and Sufficient Conditions for the Prefix Property

It should be clear from the discussion thus far that arboraceous demand graphs can be analyzed like rectilinear demand graphs as far as necessary and sufficient conditions for the prefix property are concerned. For the frustula of $F(D, \gamma_I, \gamma)$ correspond to the chains intersecting γ in a rectilinear demand graph, the demand on a sliver of a frustulum corresponds to the demand on an arc of a chain in a rectilinear graph, and so on.

Thus, the results in Lemmas 3.1 to 3.3 and Theorem 3.1 can be translated directly for arboraceous demand graphs. They are stated below as Lemmas 4.1 to 4.3 and Theorem 4.1, respectively. The proofs are similar to those in Chapter 3 and only the variations will be explained. In general, the proofs of Chapter 3 apply with substitution of "frustulum" for "chain" when the reference in Chapter 3 is to the part of a chain above a slice, and "chain-graph" for "chain" when the part referred to lies below a slice, of "sliver" for "arc","move down a chain-graph" for "move down a chain", etc., where appropriate. The notation "$\gamma \square f_i$" stands for the sliver in which γ intersects the frustulum f_i of some frustum, and $d(\gamma \square f_i)$ for the demand on that sliver.

LEMMA 4.1 Let D be an arboraceous demand graph and let γ be a feasible slice of D that intersects at least one frustulum of $F(D, \sigma, \gamma)$ in a sliver with non-zero demand, where σ is a feasible slice of D that is earlier than γ and from which γ is accessible. Let D^* be the extension of D with respect to γ defined by Figure 4.5, and δ_1 be any slice of D that is of the form F_1, which is defined below. Then the slice γ possesses the prefix property with respect to σ only if whenever the slice δ_1 is accessible from σ, the slice γ is not accessible from δ_1.

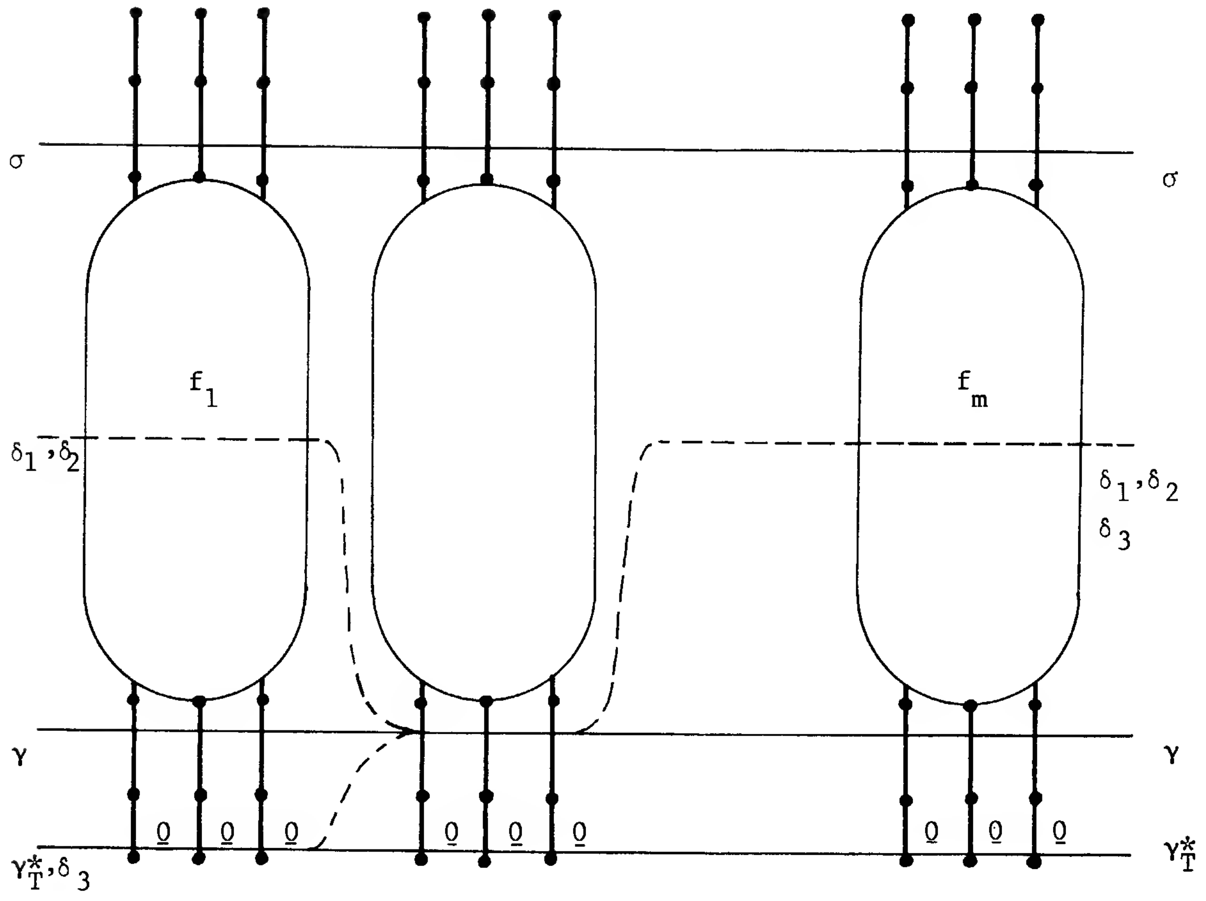
Form F_1 A slice, δ_1, of this form satisfies the following conditions:

(i) $\sigma \preccurlyeq \delta_1 \prec \gamma$

(ii) δ_1 and γ share at least one arc that has a non-zero demand

(iii) $\underline{d}(\gamma) \nleq \underline{d}(\delta_1)$

COMMENT It will be recalled that the proof of Lemma 3.1 involves constructing an extension in which σ is safe but γ is not. This is done by following $\gamma \sqcap \chi_j$, where χ_j is the chain on which γ and δ_1 share an arc, by an arc α', whose demand is just small enough for $(\alpha'/\delta_1 \sqcap \chi_j)\delta_1$ to be feasible. The arcs on $\chi_k (k \neq j)$ following γ have demands in D' which are such that uni-chain macro-moves down the χ_k's fit $(\alpha_T^j/\gamma \sqcap \chi_j)\gamma$ feasibly for some ordering of the k's, where



The Extension D*

Figure 4.5

α_T^j is the terminal arc of χ_j in D'. Thus γ_T', the terminal slice of D' is accessible from δ_1 but not γ.

In the case of arboraceous demand graphs D' is similary constructed with chain-graph read for chain. Accessible slivers of frustula play the same role as arcs that are not barriers in rectilinear graphs.

<u>LEMMA 4.2</u> Let D be an arboraceous demand graph and let γ be a feasible slice of D that intersects at least one frustulum of $F(D, \sigma, \gamma)$ in a sliver with non-zero demand, where σ is a feasible slice of D that is earlier than γ and from which γ is accessible. Let D^* be the extension of D with respect to γ defined by Figure 4.5 and δ_2 be any slice of D^* of the form F_2, which is defined below. Then the slice γ possesses the prefix property with respect to σ only if every δ_2 is inaccessible from σ.

<u>Form F_2</u> A slice, δ_2, of this form satisfies the following conditions:

(i) $\sigma \preccurlyeq \delta_2 \prec \gamma$

(ii) δ_2 and γ share at least one arc that has a non-zero demand

(iii) $\underline{d}(\gamma) \not\leq \underline{d}(\delta_2)$

(iv) $\underline{d}(\delta_2 \sqcap f_i) \leq \underline{d}(\rho \sqcap f_i)$ for all slices, ρ, which lie between σ and δ_2 (inclusive) and for all frustula f_i.

COMMENT In the case of Lemma 3.2 use is made of Lemma 3.1 and condition (iv) in F_2 is shown to imply that γ must be accessible from δ_2. The accessibility of γ from δ_2 is shown by modifying the moves from σ to γ that fit γ feasibly to fit δ_2.

In the case of arboraceous demand graphs too the moves can be modified to fit δ_2. As in the proof of Lemma 3,1, one can consider moves μ_0 and μ_1 so that the slice resulting from the application of μ_1 is the first one in the sequene of feasible slices from σ to cross δ_2, etc.

LEMMA 4.3 Let D be an arboraceous demand graph and γ be a slice of D that intersects at least one frustulum of $F(D, \sigma, \gamma)$ in a sliver with non-zero demands, where σ is a feasible slice of D that is earlier than γ and from which γ is accessible. Let D^* be the extension of D with respect to γ defined by Figure 4.5 and δ_3 be any slice of D^* that is of the form F_3, which is defined below. Then γ possesses the prefix property with respect to σ if whenever δ_3 is accessible from σ, γ_T^*, the terminal slice of D^*, is not accessible from δ_3.

Form F_3 A slice, δ_3, of this form satisfies the following conditions:

(i) $\sigma \preccurlyeq \delta_3$

(ii) Either δ_3 and γ share at least one arc that has non-zero demand or δ_3 includes at least one terminal arc of D^*.

(iii) $\underline{d}(\gamma_3) \not\leq d(\delta_3)$ where γ_3 is the slice obtained by replacing arcs in γ by terminal arcs of D^*, on all those chain-graphs of D^* defined by γ that δ_3 intersects in terminal arcs.

<u>COMMENT</u> It will be recalled that the proof of Lemma 3.3 is similar to that of Lemma 3.1, in that it involves modifying a sequence of moves from σ to γ_T', the terminal slice of an extension D' in which σ is safe, to fit γ feasibly. Exactly the same technique is applicable to arboraceous demand graphs.

<u>THEOREM 4.1</u> Let D be a vector demand graph and let γ be a feasible slice of D that intersects at least one frustulum of F(D, σ, γ) in a sliver with non-zero demand, where σ is a feasible slice of D that is earlier than γ and from which γ is accessible. Then γ possesses the prefix property with respect to σ if

$\underline{d}(\gamma \sqcap f_i) \leq \underline{d}(\rho \sqcap f_i)$ for all slices, ρ, that lie between σ and γ (inclusive) and for all frustula, f_i.

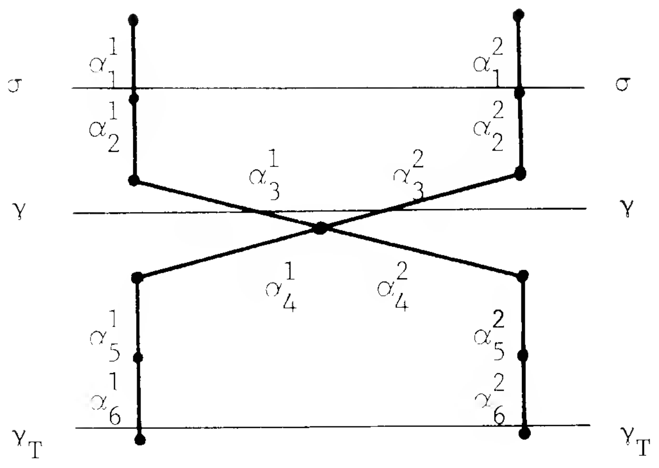


Figure 4.6a

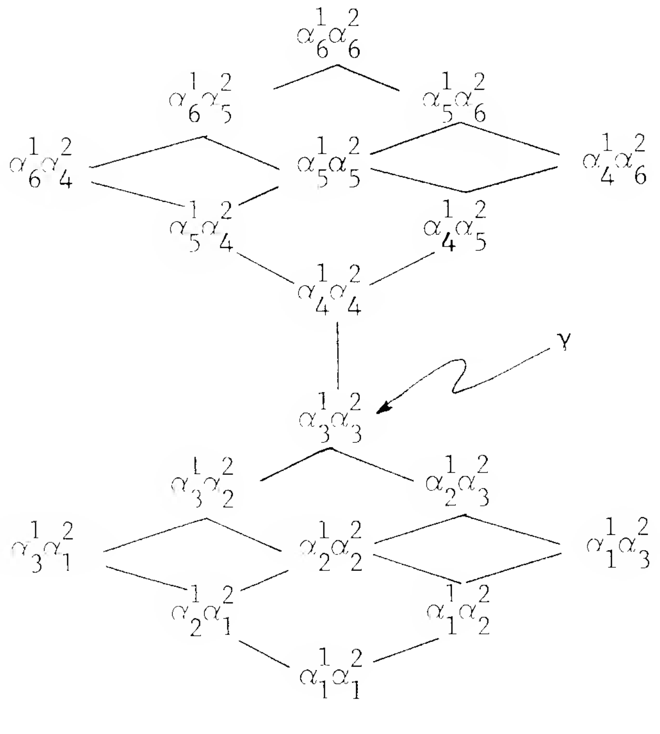


Figure 4.6b

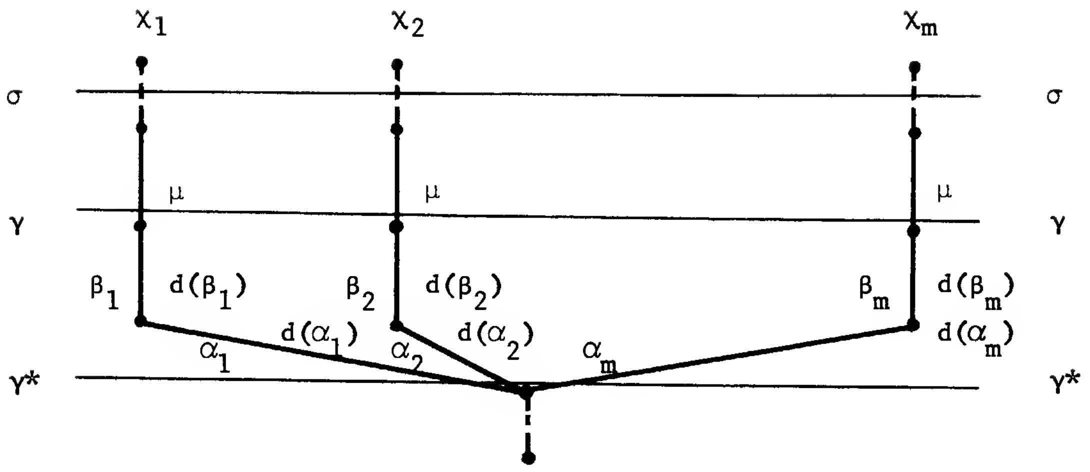
for such graphs cannot be linear. To prove this would merely require translation of Lemma 3.4 and Theorem 3.3. In fact, even translation is unnecessary as rectilinear demand graphs are special cases of arboraceous demand graphs.

However, even with scalar demands arboraceous demand graphs can have only non-linear limited-backtracking algorithms. This is proved in Theorem 4.2 below. The term "crutch" and "barrier" may be applied to slivers in addition to arcs in the rest of this chapter (although "barriers" are usually arcs), as the slivers of interest consist of single arcs in those instances.

THEOREM 4.2 There does not exist a linear limited-backtracking algorithm for arboraceous demand graphs even when the demands and capacity are chosen from the set of integers.

PROOF: Consider the demand graph in Figure 4.7. Suppose one has constructed a partial connected sequence of feasible slices from σ to γ and suppose γ possesses the prefix property with respect to σ.

Because of the choice of values for the demands associated with the α's and β's, each arc labelled α or β has a demand that is greater than the demand on the arc in γ that lies on the same frustulum of $F(D, \gamma, \gamma^*)$. No slice, γ', that lies strictly between γ and γ^* can



$d(\alpha_i) = \mu + i \ ; \ i \in [1,m]$

Capacity $= m + m.\mu$

$d(\beta_i) = \mu + (m-i+1) \ ; \ i \in [1,m]$

Figure 4.7a

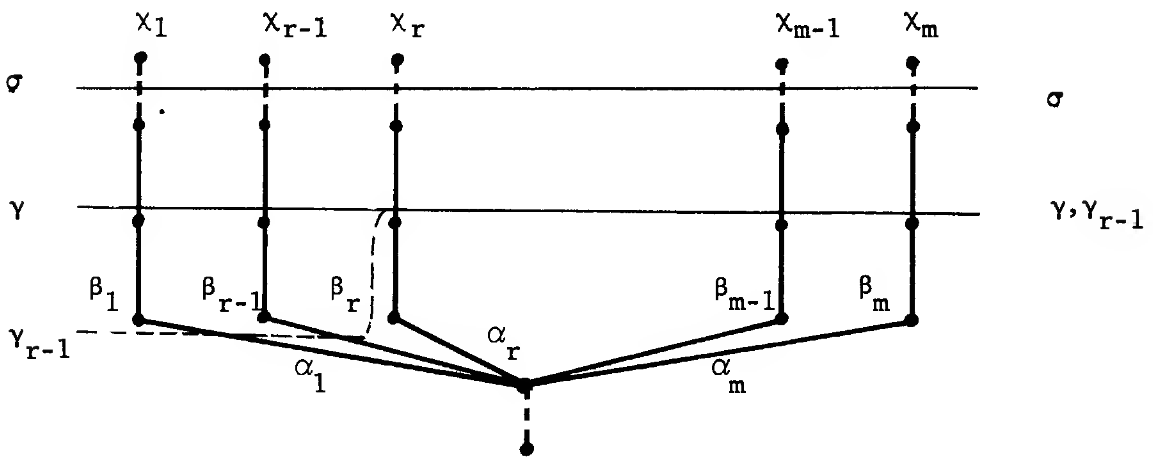


Figure 4.7b

possess the prefix property with respect to γ. For the macro-move $\gamma \to \gamma'$ can be broken up into exactly m uni-chain macro-moves $\mu_1, \mu_2, \ldots \mu_m$ because of the relations between demand values indicated in the previous sentence, and the slice $\gamma\mu_1$ is a slice that satisfies all the conditions of Form F_1 in Lemma 4.1.

Thus the next slice that possesses the prefix property with respect to γ lies below γ^*.

Careful observation of the figure shows that there is exactly one ordering of the chain-graphs $\chi_1, \chi_2, \ldots \chi_m$ defined by γ for the uni-chain macro-moves making up the macro-move $\gamma \to \gamma^*$ that fits γ. This order is $\chi_1, \chi_2, \ldots \chi_m$ in the figure but can be made aribtrary by permuting the values of demand on the α_i's and β_i's.

As there is no way in which a local algorithm can determine the one order that is correct, other than by trial and error, the number of futile trials, consisting of uni-chain macro-moves, can be (conservatively speaking) as large as

$$(m - r) + (m - r) + \ldots m - r \text{ times} = (m - r)^2$$

For each of the m - r uni-chain macro-moves, $\mu_j = \gamma_{r-1} \to (\alpha_j/\gamma_{r-1} \sqcup \chi_j)\gamma_{r-1}$ for the values of j in [r, m], fit γ_{r-1}. Of these all but one are incorrect. However, that the macro-move $\mu_j (j \neq r)$ is incorrect is not discovered until m - r futile uni-chain macro-moves (down $\chi_r, \chi_{r+1}, \ldots \chi_{j-1}, \chi_{j+1}, \ldots \chi_m$) are tried from $\gamma_{r-1}\mu_j$.

Thus as many as

$$(m - 1)^2 + (m - 2)^2 + \ldots + 1^2 = \frac{1}{6}(m - 1)(m)(2m - 1)$$

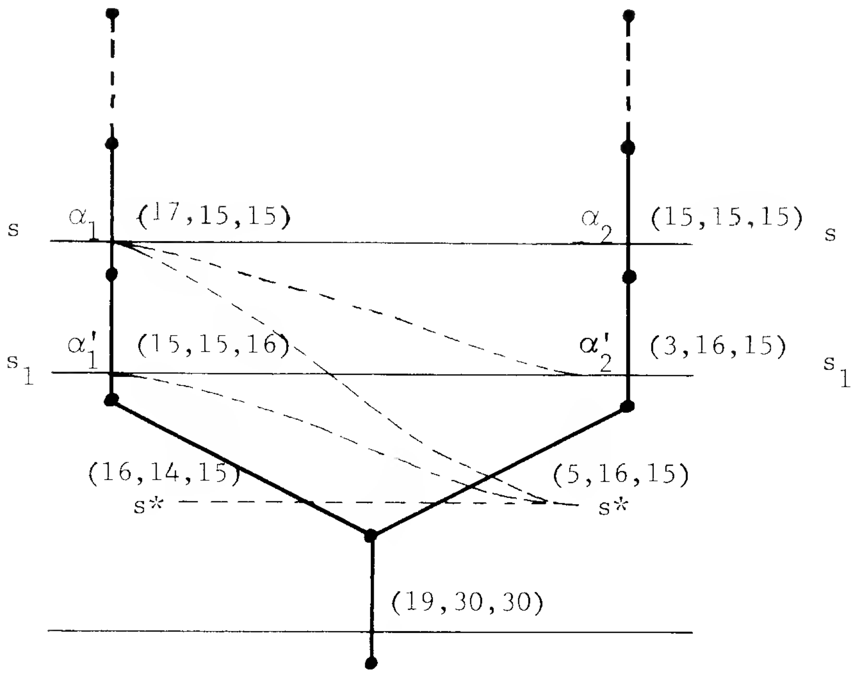
trials can be wasted.

The non-linearity of any limited-backtracking algorithm follows, since there is always a graph that has a "correct order" different from that used by the algorithm, andin fact a correct order that is as bad as the worst.

Q.E.D.

§4.9 On the Non-local Nature of Algorithms for Arboraceous Demand Graphs

Consider the frustulum shown in Figure 4.8. Two slivers s and s^* are shown there. Suppose σ is a feasible slice which contains s and γ is the feasible slice $(s^*/s)\sigma$. The slice γ does not possess the prefix property with respect to σ because the slice $(s_1/s)\sigma$ is a slice of the form F_1 in Lemma 4.1. However, if a safeness algorithm were to use the macro-move $\sigma \to \gamma$ shown by the sequence of dashed slivers in Figure 4.8, then $(s_1/s)\sigma$ is not a slice that is part of the connected sequence $\sigma \ldots \gamma$. That a general limited backtracking algorithm needs information about slices that are not in the sequence of slices it constructs to determine whether a macro-move is acceptable or not, means that such an algorithm is not local in the defined sense.



'Capacity' = (100,100,100)

Figure 4.8

It may be local in the broader sense that it uses only the initial part of the demand graph up to the slice reached.

The problem is not restricted to demand graphs of which the frustulum in Figure 4.8 is a part. Rather, it is a consequence of two facts. The first fact is that there are more slices in a frustulum which is followed by a point of synchronisation than in a sequence of slices that is produced by a macro-move that crosses it. The other fact is that crutches such as those in s_1 can lead to slivers of smaller demand in combination with other crutches. For instance, in Figure 4.8, the sliver s^* has a demand no greater than that of any of the dashed slivers encountered, andyet this is not so with respect to s_1. Thus a translated version of the Modified Safeness Algorithm of Chapter 3 would have (erroneously) declared the macro-move $\sigma \rightarrow \gamma$ acceptable! However, this version of the Modified Safeness Algorithm would not be in error in this manner if the demands were scalar; for if

$$a' \nleq a \quad \text{and} \quad b' \nleq b$$

then

$$a' + b' \nleq a' + b'$$

which is not necessarily true for vectors, as the arcs α_1, α_1' and α_2, α_2' show in Figure 4.8. The sliver s_1 has a smaller demand than s does and also a smaller demand than $(\alpha_2'/\alpha_2)s$ or $(\alpha_1'/\alpha_1)s$ do.

Let γ_s be a slice that uses the sliver s and γ_s^* a slice that uses s^*. Then the fact, that all the feasible slices that are

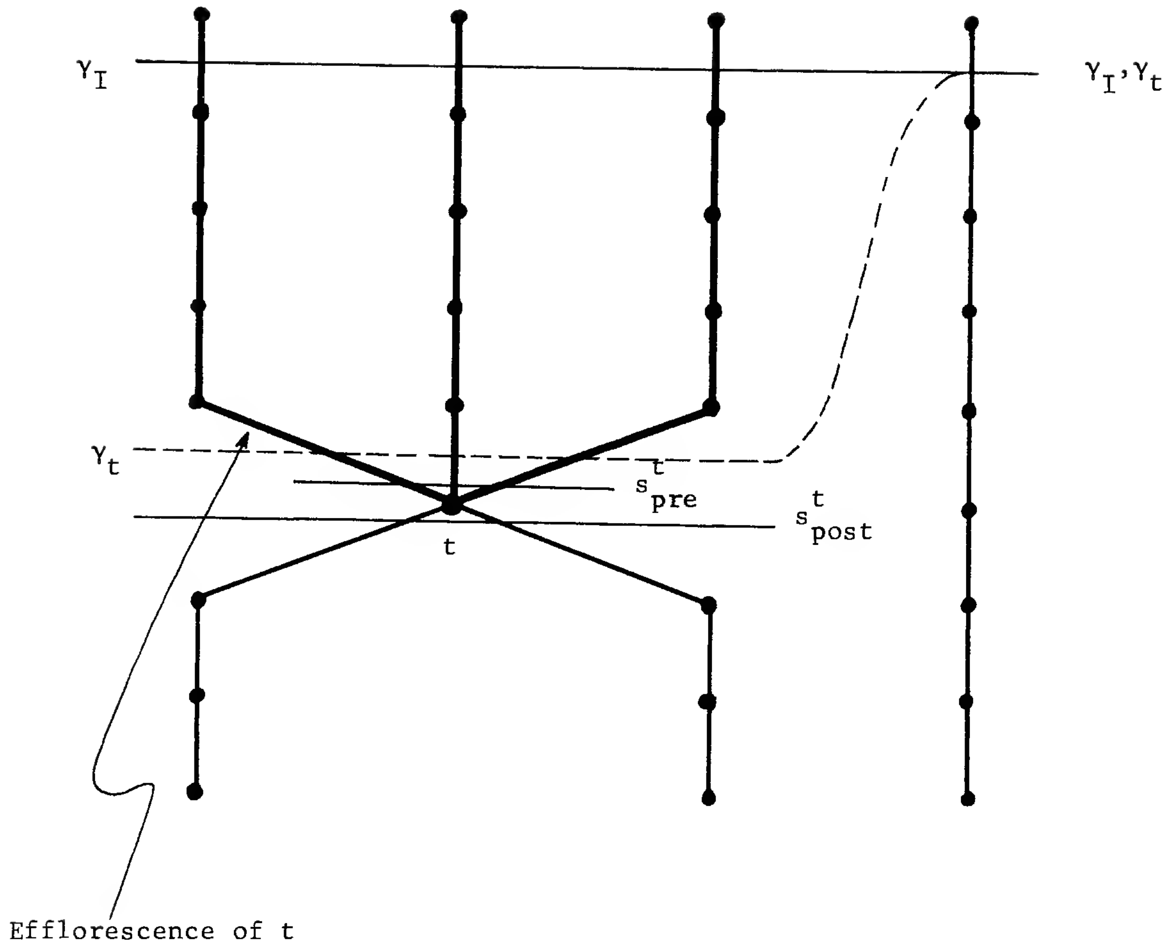

γ_I
γ_I, γ_t
γ_t
s^t_{pre}
s^t_{post}
t
Efflorescence of t

Figure 4.9

accessible from γ_s and that lie betweeen γ_s and γ_s^* need to be considered if Theorem 4.1 is to be applied in an algorithmic test, is crucial to the understanding of the General Safeness Algorithm. This algorithm is presented in the next section. Fortunately, s need not be earlier than the last sliver that meets the test of Theorem 4.1, relative to the corresponding sliver in the test slice, on each of the chain-graphs that join at the point of synchronisation.

§4.10 The General Safeness Algorithm

The General Safeness Algorithm, or GSA for brevity, is an algorithmic test for testing the safeness of a slice of an arboraceous demand graph. It attempts to construct a connected sequence of feasible slices from the test slice to the terminal slice of the demand graph. Some new terminology is useful in the description of the GSA and is indicated below.

The pre-synchronisation sliver, s^t_{pre}, of a point of synchronisation, t, is the sliver that contains exactly those arcs which are the incoming arcs of t. Similarly, the post-synchronisation sliver, s^t_{post}, of t is the sliver that contains exactly those arcs which are the outgoing arcs of t. Figure 4.9 shows the pre-synchronisation sliver and post-synchronisation sliver of a point of synchronisation.

The efflorescence $\mathcal{E}(t)$ of a point of synchronisation, t, is the frustulum of $F(D, \gamma_I, \gamma_t)$ that contains the arcs incident on t — where

FUNDAMENTAL ALGORITHM

FA is similar to the Basic Algorithm of Chapter 3. The input parameters ω and γ are respectively the reference slice and the current conditionally acceptable slice. In case of successful termination, FA returns a slice, γ_p, that possesses the prefix property with respect to ω. Nothing is returned in the event of failure.

The set X'_{FA} is in internal variable. The set S is an input parameter and a set of slices that are relevant to the application of the test in Theorem 4.1. S_t is a similar set except that it is of temporary interest and is an internal variable. X_{FA} is an input parameter and is a set of chain graphs.

Step 0: Set S_t equal to Φ, the empty set, and X'_{FA} equal to X_{FA}. Go to Step 1.

Step 1: Add γ to S_t. Go to Step 2.

Step 2: Pick a chain-graph from X_{FA} - call it χ_i. Go to Step 3.

Step 3: Attempt to construct a uni-chain macro-move, μ, down χ_i that fits γ and is as large as possible, but terminate the macro-move at the first point where the slice γ' resulting from the application of μ satisfies one of the conditions given below. In any case, add the slices resulting from the component moves of μ to the set S_t.

(i) $\underline{d}(\gamma' \boxtimes \chi_i) \le \underline{d}(\rho \boxtimes \chi_i)$ for all slices ρ that lie between γ and γ' (inclusive)

Go to Step 4.

(ii) γ' is following by a fork, f,

In this case perform the Fork Algorithm (FkA, for brevity) with γ', X_{FA} and f as values for the input parameters γ_F, X_F and f_F.

If FkA terminates with failure, go to Step 5.

If FkA terminates successfully, set γ and X_{FA} respectively equal to γ_F^* and X_F^*, the values returned, and go to Step 4.

(iii) γ' is followed by a point of synchronisation, t.

In this case, go to Step 5 after setting S_t to Φ.

<u>Step 4</u>: If γ' satisfies:

$\underline{d}(\gamma' \sqcap f_i) \leq \underline{d}(\rho \sqcap f_i)$ for all slices ρ in S and for all frustules, f_i, of $F(D, \omega, \gamma')$

then set γ_p equal to γ', terminate and report success.

If γ' does not satisfy the above condition then add S_t to S, set γ equal to γ' and both X'_{FA} and X_{FA} equal to the set of chain-graphs defined by γ', and go to Step 2.

<u>Step 5</u>: Delete χ_i from X'_{FA}. If X'_{FA} is now empty then go to Step 6; if not, go to Step 1.

<u>Step 6</u>: Perform the Sync Algorithm (SA) with γ, ω, X_{FA} and S as respective values for the input parameters, γ_{SA}, ω_{SA}, X_{SA} and

S_{SA}. If SA terminates successfully, set γ' equal to γ^*_{SA}, the value returned, and go to Step 4. If SA terminates with failure go to Step 7.

<u>Step</u> <u>7</u>: Perform the Crutch Algorithm (CA) with γ, ω, X_{FA} and S as respective values for the input parameters γ_{CA}, ω_{CA}, X_{CA} and S_{CA}. If CA terminates successfully, set γ equal to γ^*_{CA}, the value returned, and go to Step 4. If CA terminates with failure, terminate and report failure.

<u>SYNC ALGORITHM</u>

Input parameters to this algorithm are γ_{SA}, ω_{SA}, X_{SA} and S_{SA}. The algorithm searches the chain-graphs in X_{SA} one at a time until a point of synchronisation is reached. If it finds such a point, t, it seeks the aid of the Sync Crosser Algorithm (SCA) to extend the sequence from γ_{SA} to a post-synchronisation slice of t and (recursively) asks for the performance of FA. The parameter ω_{SA} is a slice. X'_{SA} is an internal variable and is initialized to X_{SA}. S_{SA} is a set of slices.

<u>Step</u> <u>0</u>: Set X'_{SA} equal to X_{SA} and go to Step 1.

<u>Step</u> <u>1</u>: Pick a chain from X'_{SA} — call it χ_i. Go to Step 2.

<u>Step</u> <u>2</u>: Attempt to construct a uni-chain macro-move, μ, down χ_i that fits γ_{SA} and that is such that $\gamma_{SA}\mu$ is followed by a point of synchronisation, t.

If the attempt is successful go to Step 3; if not go to Step 4.

<u>Step 3</u>: Perform the Sync Crosser Algorithm (SCA) with $\gamma_{SA}\mu$, X_{SA} and t as values for γ_{SCA} X_{SCA} and t_{SCA}, respectively. If SCA terminates successfully, set γ_{SA} and X_{SA} respectively equal to γ^*_{SCA} and X^*_{SCA}, the values returned, augment S_{SCA} with S^*_{SCA} which is returned, and go to Step 5.
If SCA terminates with failure, go to Step 4.

<u>Step 4</u>: Delete χ_i from X'_{SA}. If X'_{SA} is now empty go to Step 6; if not, go to Step 1.

<u>Step 5</u>: Perform FA with ω_{SA}, γ_{SA}, X_{SA} and S_{SA} as respective values for ω, γ, X_{FA} and S.
If FA terminate successfully, set γ^*_{SA} equal to γ_p, the value returned, terminate and report success.
If FA terminates with failure, go to Step 4.

<u>Step 6</u>: Terminate and report failure.

<u>Sync Crosser Algorithm</u>

This algorithm uses the Enumerative Algorithm (EA) to build up a set, S^*_{SCA} of slices that are feasible and accessible from, γ_{SCA}, one of its input parameters and determines if the pre-synchronisation sliver s^t_{pre} of, t_{SCA}, another input parameter is accessible from γ_{SCA}. The parameter X_{SCA} is a set of chain-graphs as is X^*_{SCA}, but the former is an input parameter and the latter is returned upon successful termination. A slice, γ^*_{SCA}, and the set S^*_{SCA} of slices are also returned upon successful termination.

Step 1: Construct the set of chain-graphs in the efflorescence of t_{SCA}. Call it X'_{SCA}. (At worst, X'_{SCA} can be set equal to X_{SCA}.) Go to Step 2.

Step 2: Perform the Enumerative Algorithm with X'_{SCA} and γ_{SCA} as respective values for the input parameters.

If EA terminate successfully, set S^*_{SCA}, γ^*_{SCA} and X^*_{SCA}, respectively equal to the values S^*_{EA}, γ^*_{EA} and X^*_{EA} returned, terminate report success.

If EA terminates with failure, terminate and report failure.

Enumerative Algorithm

This is a recursive algorithm similar to the Crutch Algorithm of Chapter 3, except that it asks for the performance of EA instead of the Basic Algorithm and that it needs to use FkA at forks and to treat points of synchronisation as barriers. It builds up the set S^*_{EA} of feasible slices accessible from γ_{EA} and terminates with success if $(s^t_{pre}/s_{\gamma_{EA}})\gamma_{EA}$, the slice that is identical to γ_{EA} except that it uses the pre-synchronisation sliver, is in S^*_{EA}. Upon successful termination S^*_{EA} is returned, as is γ^*_{EA}, which is $(s^t_{post}/s)\gamma_{EA}$, and X^*_{EA}, the set of chain-graphs defined by γ^*_{EA}.

CRUTCH ALGORITHM

This algorithm is similar to its namesake in Chapter 3, except that it uses FkA when necessary and seeks performance of FA instead of the Basic Algorithm.

FORK ALGORITHM

It takes three input parameters, a slice γ_F, a set of chains X_F, and a fork f_F. If the slice through the post-fork sliver (this is similar to the post-synchronisation sliver, conceptually) i.e., $(s^f_{post}/s_{\gamma_F})\gamma_F$ is feasible, it terminates with success and returns this slice as γ^*_F and the chain-graphs it defines as X^*_F. No value is returned if FkA fails.

§4.11 Isolation of Efflorescences

The SCA algorithm in the previous section assumed that the efflorescence of a point of synchronisation can be isolated. The task is far from easy as Figure 4.10 shows. In Figure 4.10, if the chain-graph of γ were searched from top to bottom to determine if t lies on them, a fairly long and futile search down the chain-graphs marked χ_f and χ_f' is possible before it is realized that t is not on it. Besides, unless the points of synchronisation are labelled too, there

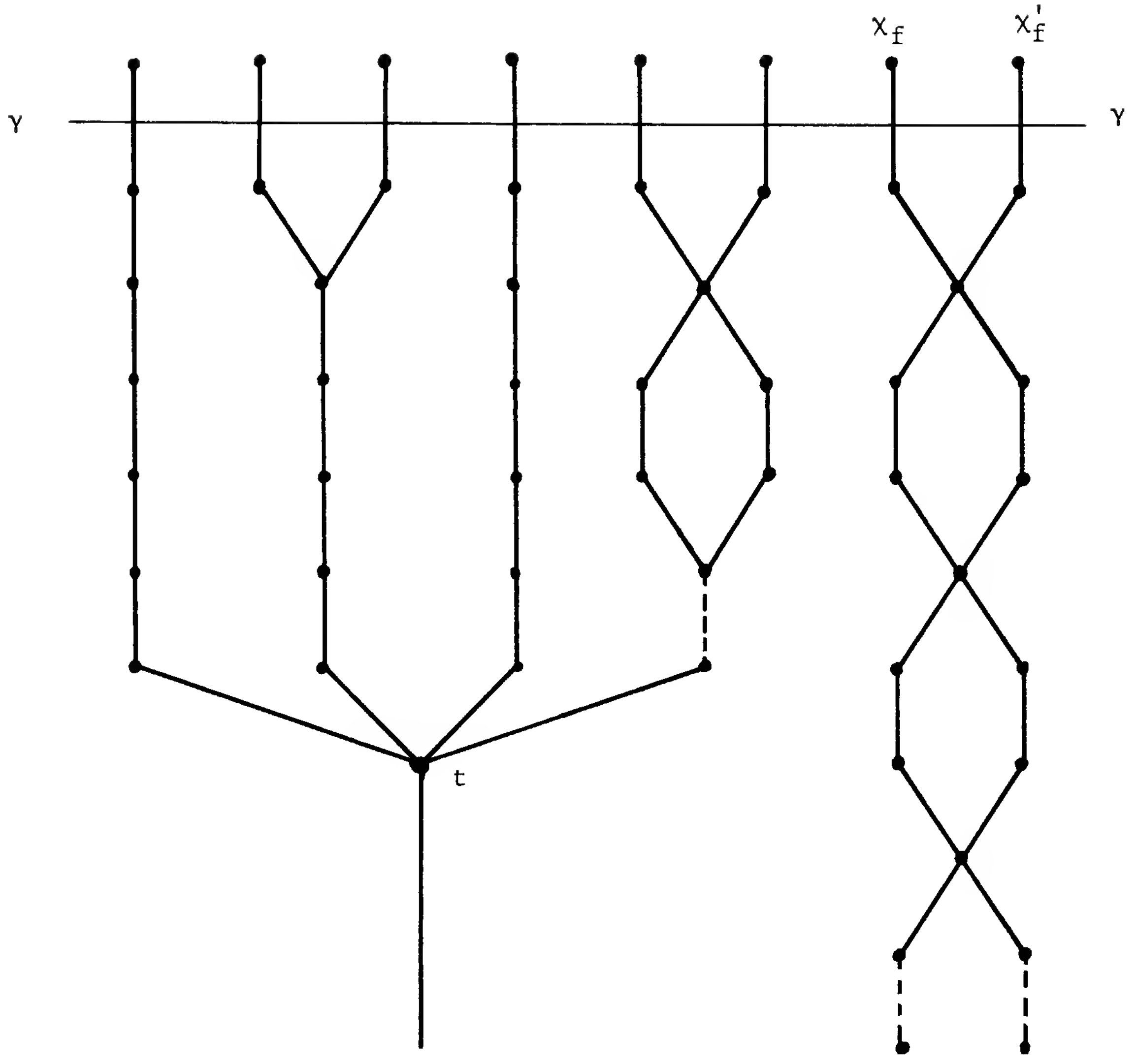


Figure 4.10

is no way of distinguishing one from another.

The isolation of an efflorescence becomes considerably easier if the demand graphs are constrained so that graphs such as that in Figure 4.11 are ruled out; for them the chain-graphs can be labelled conveniently. The constraint can be described precisely if the notion of generations is associated with chain-graphs. For this purpose it is useful to use chains again. A chain-graph of γ_I starts out as a chain and sub-divides into more chains, with consolidation occurring at some points of synchronisation. Points of synchronisation will be referred to as joins.

The first constraint requires that all points of synchronisation have exactly one outgoing arc. It will be recalled that forks have exactly one incoming arc. This makes it possible to introduce the concept of generation.

The chains that are chain-graphs defined by γ_I belong to the first generation. At a fork, such a chain gives rise to two or more chains of the second chain. Each chain of the second generation gives rise to chains belonging to the third generation at a fork, and so on. Similarly, chains give rise to a chain of one lower generation at a join. However, this leads to an ambiguity if chains of different generation meet at a join. The second constraint, therefore, requires that only chains belonging to the same generation can meet at a join.

Chains of second or older generations that arise from a chain,

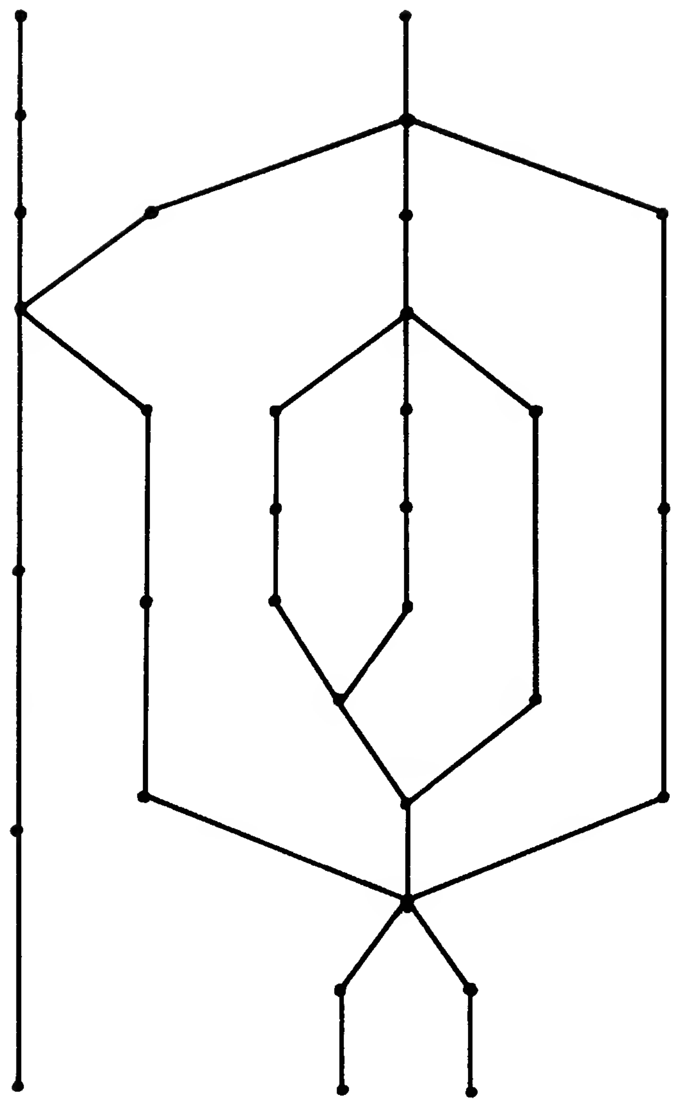


Figure 4.11

that is a chain-graph defined by γ_I, are called siblings. All chains of the first generation are also siblings.

The third constraint requires all chains meeting at a join to be siblings in addition to belonging to the same generation.

These three constraints are necessary for consistency of generation numbering.

Figure 4.12 shows arcs marked with the generations of the chains they belong to. The demand graph of Figure 4.12 satisfies all the constraints.

Figure 4.13 is a copy of Figure 4.11 but shows a one-digit position per generation labelling with increasing numbers from left to right on outgoing arcs of a fork. It is seen that the efflorescence of t consists of all chain-graphs that are labelled with a leading 1.

The constraints described above have a meaningful interpretation in terms of processes in a computer system. They state that processes are created by a computation to carry out an internal computation and, therefore, no other computation knows about the processes. A similar argument is used for processes of the third generation, and so on. Since only processes of the same generation that are siblings "know each other", only they can interact. The constraint on points of synchronisation that they have only one outgoing arc is a relatively artificial constraint, though. However, it does simplify the task of isolating efflorescences.

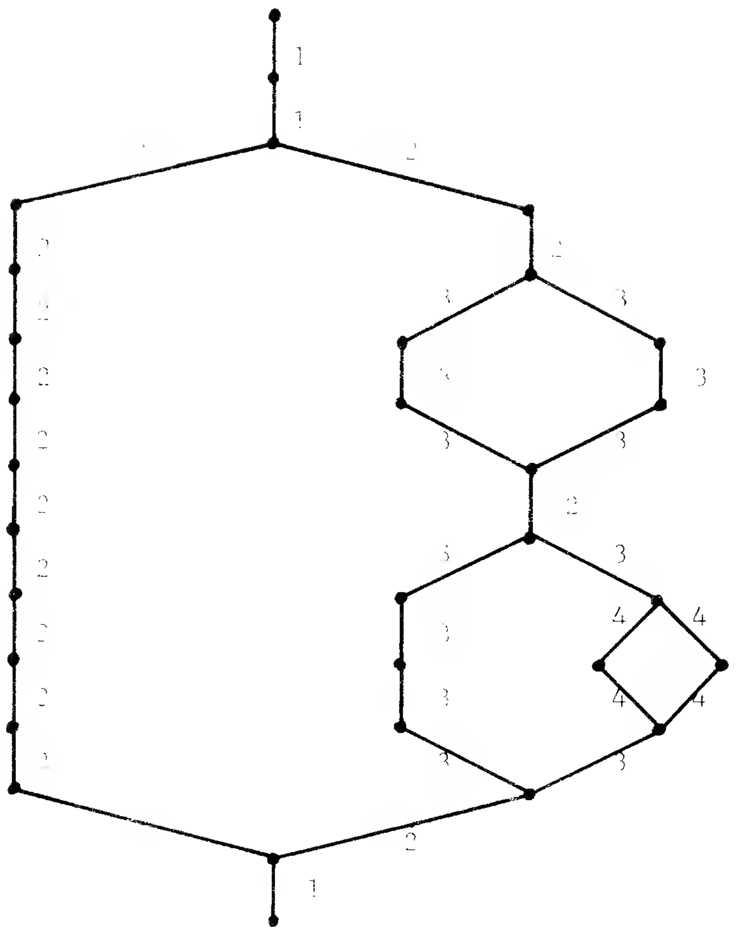


Figure 4.12

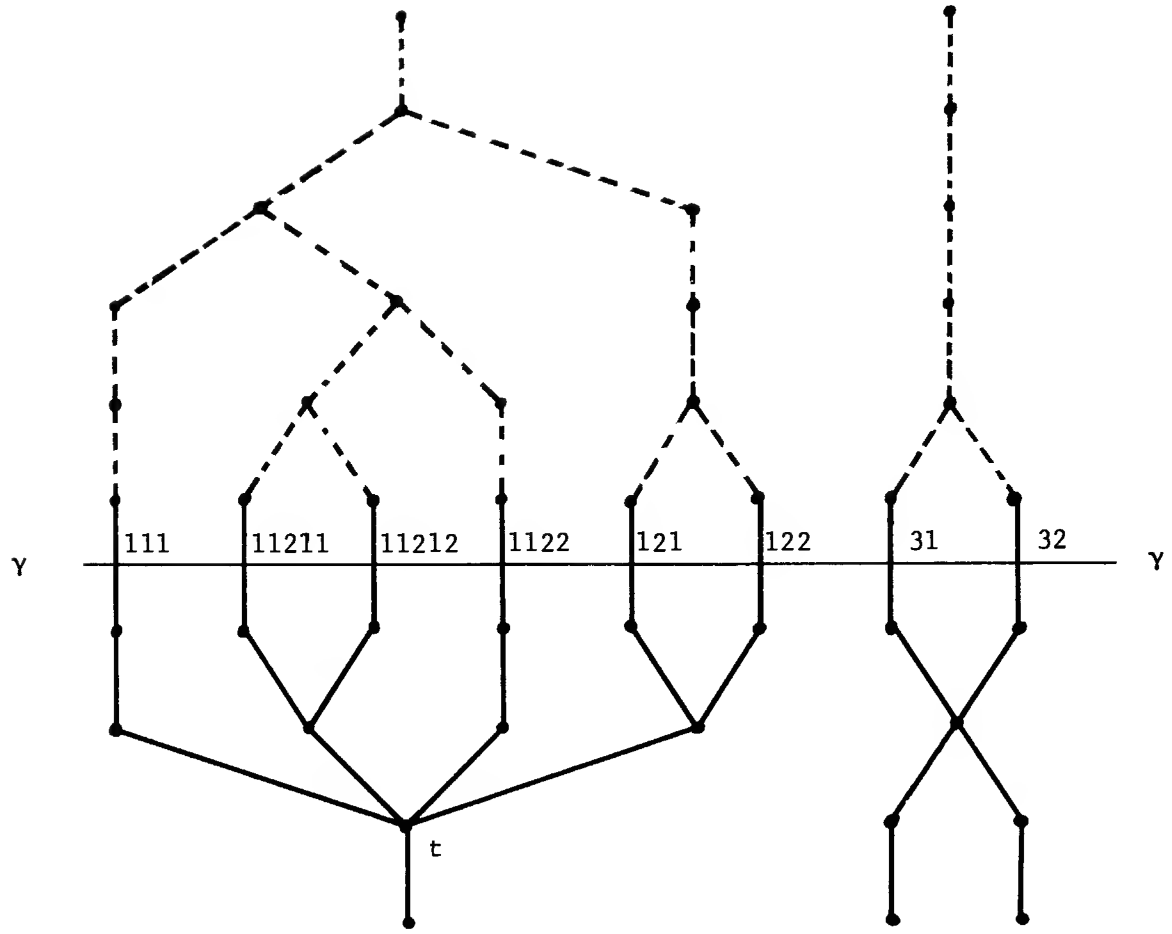


Figure 4.13

Loops and Decisions

Chapter 5

§5.1 Unrestricted and Augmented Demand Graphs

A demand graph was defined in Chapter 2 to be a finite directed graph with demands on the arcs and a capacity associated with the graph. The analysis in Chapter 4 dealt with all but demand graphs with circuits or directed cycles. Sections 5.2 and 5.3 aim at an informal study of the effect of cycles in demand graphs on the analysis of deadlocks. The study is informal because the complexity of the graphs to be considered becomes unmanageable. Moreover, the analysis of Chapter 4 suggests that there can be much repetition of familiar techniques, so that an analysis of the differences alone may suffice. Section 5.4 deals with augmentation of demand graphs to include a mechanism for the representation of decisions and alternative allocation possibilities in processes. There, too, an informal discussion of the effect of such augmentation on the analysis is presented. Because the discussion is informal, there is an underlying assumption in all sections that the demand graphs that should be considered are those that represent meaningful behaviour by users of systems, rather than general members of the classes of graphs considered.

§5.2 Unrestricted Demand Graphs

Unrestricted demand graphs are the demand graphs defined in Section 2.2, and thus include cyclic graphs. However, rather than treat such graphs in general, the discussion in this section and the next deals with rectilinear demand graphs in which arcs have been added for the purpose of creating cycles. Figure 5.1 shows an example of such a graph.

The demand graph of Figure 5.1 exhibits overall loops, i.e. it is a rectilinear demand graph in which the corresponding terminal arcs and initial arcs of chains are joined. The graphs thus consist of chains and rings. Demand graphs with overall loops will be referred to as annular demand graphs.

In terms of systems of processes, annular demand graphs represent repeatable or recurrent processes. The manufacturing industry provides several instances of recurring processes in the field of operations research. In interactive computer systems, a process that responds to editing commands or a process that handles console commands is an example of a recurrent process.

It is clear that slices of annular demand graphs can be defined, exactly as in Chapter 2, as sets of arcs, one from each chain. However, the slices do not form a lattice as they did in Chapter 2, since $\gamma_1 \preccurlyeq \gamma_2$ and $\gamma_2 \preccurlyeq \gamma_1$ do not necessarily imply that $\gamma_1 = \gamma_2$. Feasibility and safeness of a slice can be defined as before. However, as Figure 5.1 shows, if a slice such as γ is safe, then a slice such as γ' is safe too. For the (now merged) initial and terminal arcs have zero demand and the arcs on any chain have a demand that does not exceed the capacity of the graph. Thus annular demand graphs may be analyzed by cutting each ring at any arc that has zero demand and analyzing the rectilinear demand graphs that result by the techniques of Chapter 3.

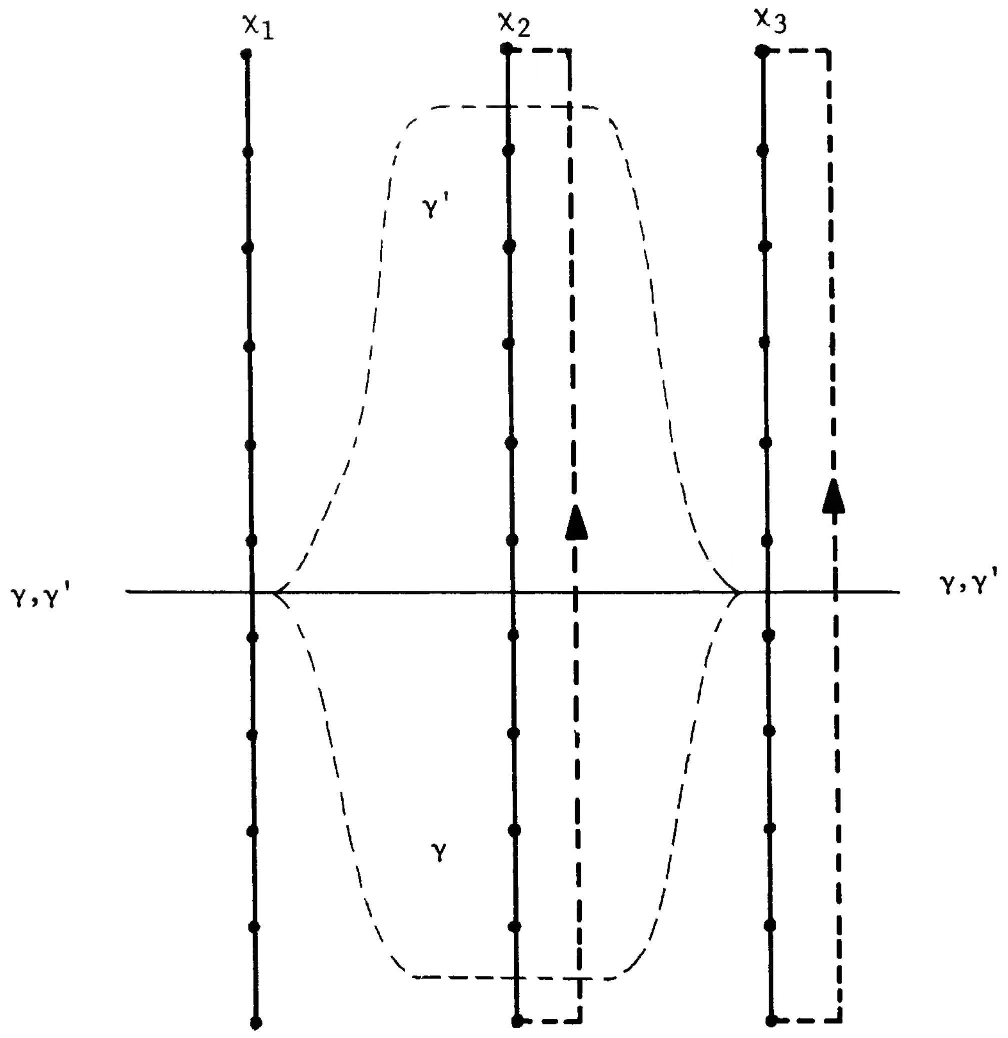
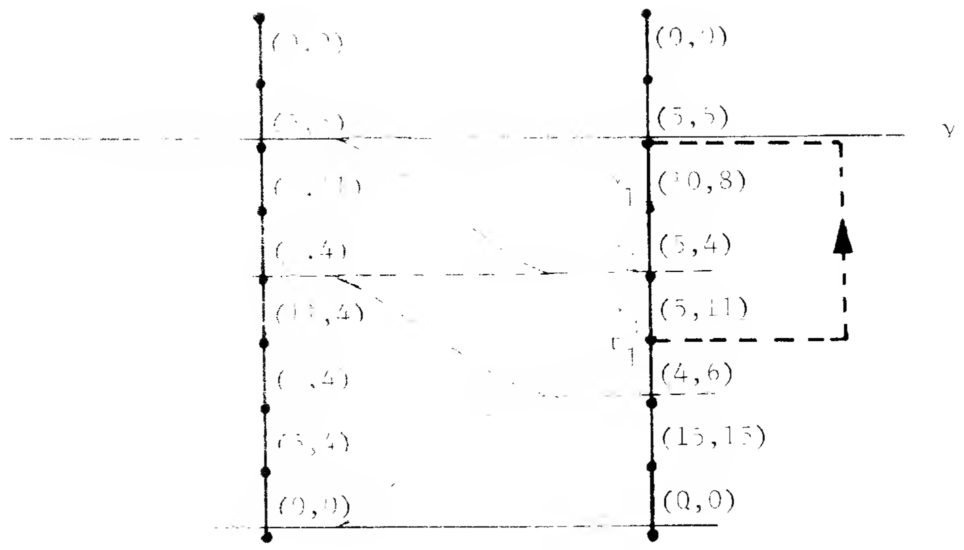


Figure 5.1

Figure 5.2 shows another form of cyclic demand-graph, viz one with an internal loop. Although moves across transitions having more than one output arc have been interpreted so far as representing the initiation of parallel processes, it is clear that such an interpretation would be meaningless for the demand graph of Figure 5.2 A useful interpretation would consider a transition, such as t_1 in Figure 5.2, which has several outgoing arcs, one of which is part of a loop, as representing a point of choice. Consequently, a slice of such a demand graph should not be defined, as it has been in Chapter 4, in terms of slivers that are cut-sets of component sub-graphs of the demand graph. Rather than attempt to find an appropriate definition of a slice for analysis of deadlock, it may be worthwhile to determine if the loops can be meaningfully rectified; for then the definitions of slices, safeness, etc., used in Chapter 3 as well as the analysis in that chapter can be used.

Now a loop in a demand graph such as that of Figure 5.2 represents the fact that the phases represented by the arcs around which the loop is drawn (the three arcs α_1, α_2, and α_3 in Figure 5.2) may occur more than once and, in fact, an unpredictable number of times. Consequently, in rectification of such a graph, it must be ensured that a slice such as γ in Figure 5.2 is considered safe only if it is safe no matter how many times the string of arcs α_1, α_2, α_3 is repeated in succession. The rectified graph used for safeness analysis must, therefore, use an adequate number of copies of the iterand, viz the



Capacity = (15, [illegible])

Figure 5.?

segment α_1, α_2, α_3. The problem of determining what number of copies is adequate will be referred to as the adequate rectification problem.

§5.3 The Adequate Rectification Problem

Consider the slice γ in Figure 5.2. If using one copy of the iterand is adequate, then Figure 5.2 shows that γ is safe. However, the result is fallacious, since it is clear that if the phases represented by the iterand do get repeated then deadlock would result in the system represented. Figures 5.3a and b show an example in which γ is safe when two copies of the iterand are used but not when three copies are.

Figure 5.4a shows a somewhat different example, in which the slice γ is safe when one or two copies of the iterand are used (Figure 5.4b) and also when any larger number of copies is used. The difference seems to lie in the fact that in Figure 5.4b one can find a slice γ', that is accessible from γ and that has the property that a uni-chain macro-move across the entire iterand fits γ' feasibly. Clearly a sequence of any number of such macro-moves across copies of the iterand would fit γ' too.

In general, let γ be the slice whose safeness is being examined. Let γ be safe when 1, 2, 3, ... n copies of the iterand are used, and let n be the smallest number such that when n copies are used, a slice γ' is accessible from which a uni-chain macro-move across the entire n^{th} copy of the iterand fits γ' feasibly. Then n is the number of

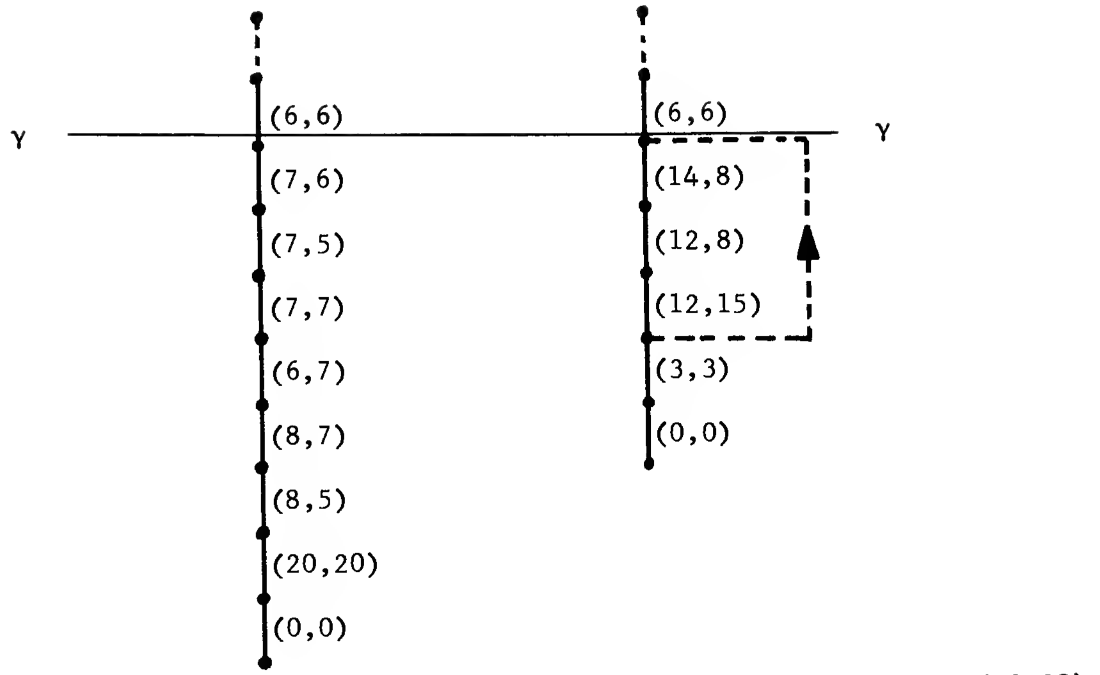


Figure 5.3a

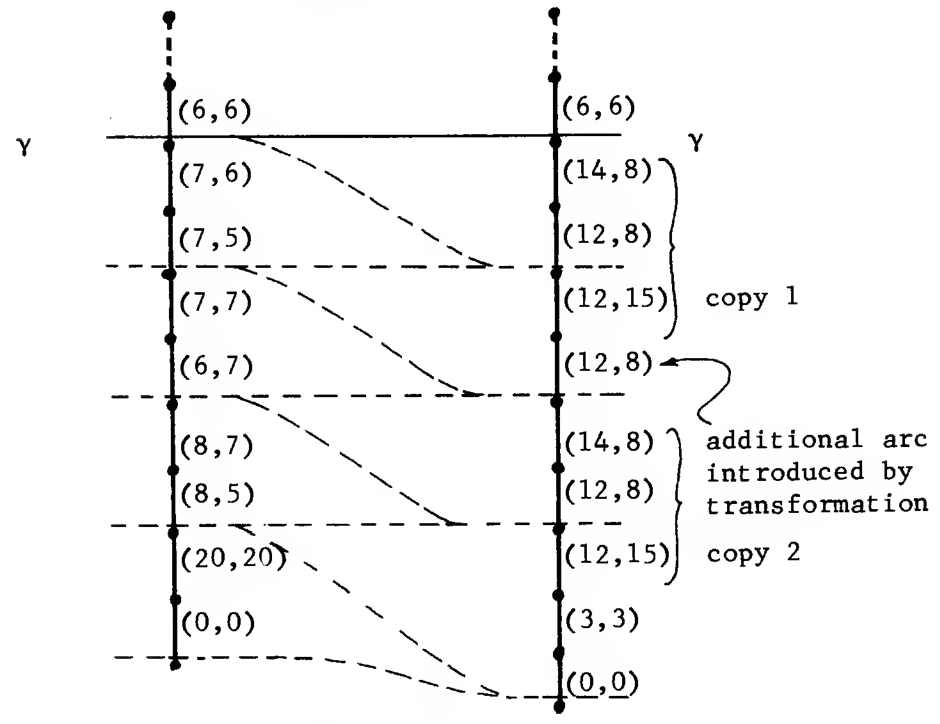


Figure 5.3b

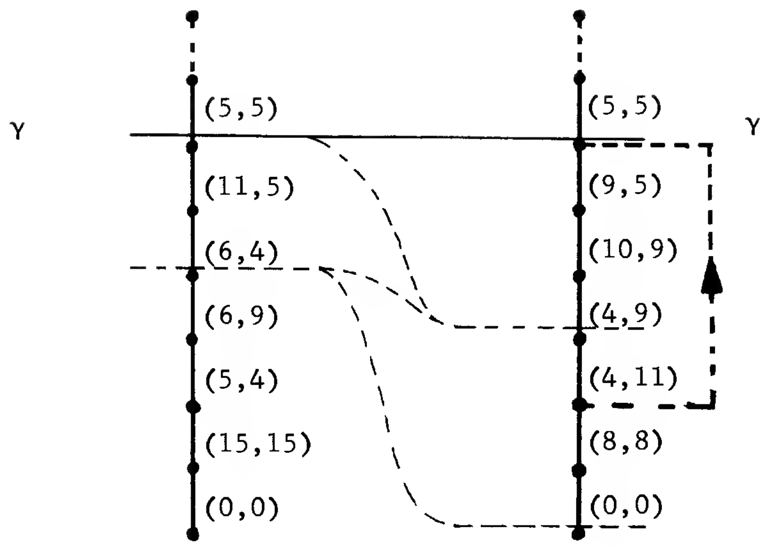


Figure 5.4a

Capacity = (15,15)

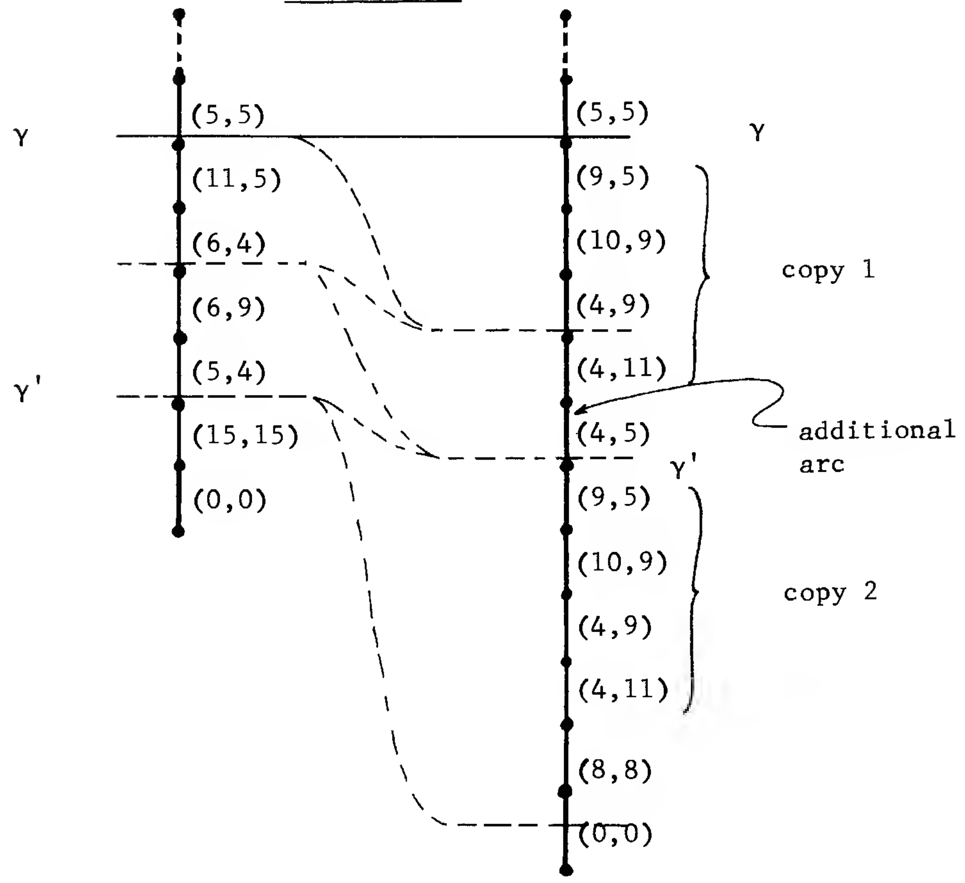


Figure 5.4b

copies that represents "adequate rectification." It would seem quite likely that n varies from test slice to test slice.

It would appear that in demand graphs with scalar demands one copy is adequate. This is because a reduction in demand cannot be selectively for one component only (as was the case with arc α in Figure 5.4a).

The number of copies referred to above is the number of complete copies — the qualification is redundant except when the test slice itself includes an arc from the iterand.

§5.4 Manifold Demand Graphs

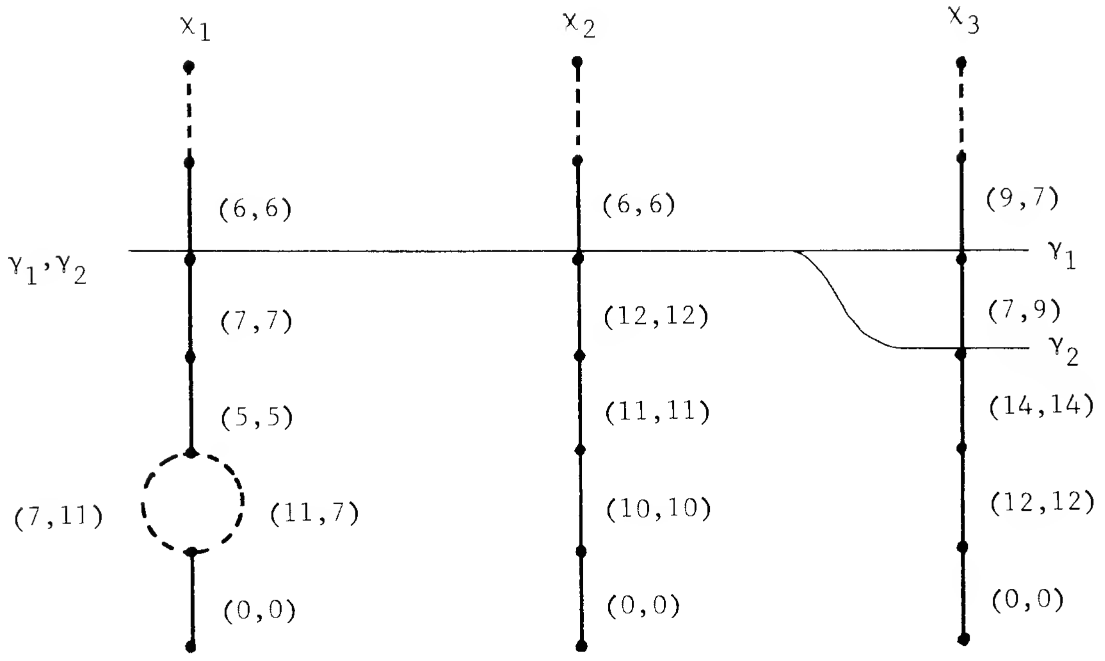
A Manifold Demand Graph is an augmented form of demand graph in which some transitions with more than one output arc are marked with the logical Exclusive Or symbol. Such transitions represent points of choice in the processes. A process takes only one of the many paths at such a point during a run. As in Section 5.2, the aim of this section is to examine the effect of such an augmentation on the analysis of deadlock and, consequently, the demand graphs considered will consist of chains.

A point of choice may arise in processes because the choice of activity to be undertaken next depends on a decision that is based on a predicate which cannot be evaluated until this point in the process. It may arise also from the presence of versatile resources in the system. Such resources can serve as well as resources of another type and, therefore, may be used in place of the latter if these are unavailable at the time.

As discussed in Section 5.2, the problem of a suitable definition for slices arises. Once again, it is tempting to try and avoid the problem by replacing the multiple chains of arcs emanating from such transitions by a single representative chain. The analysis of rectilinear demand graphs in Chapter 3 would then be applicable.

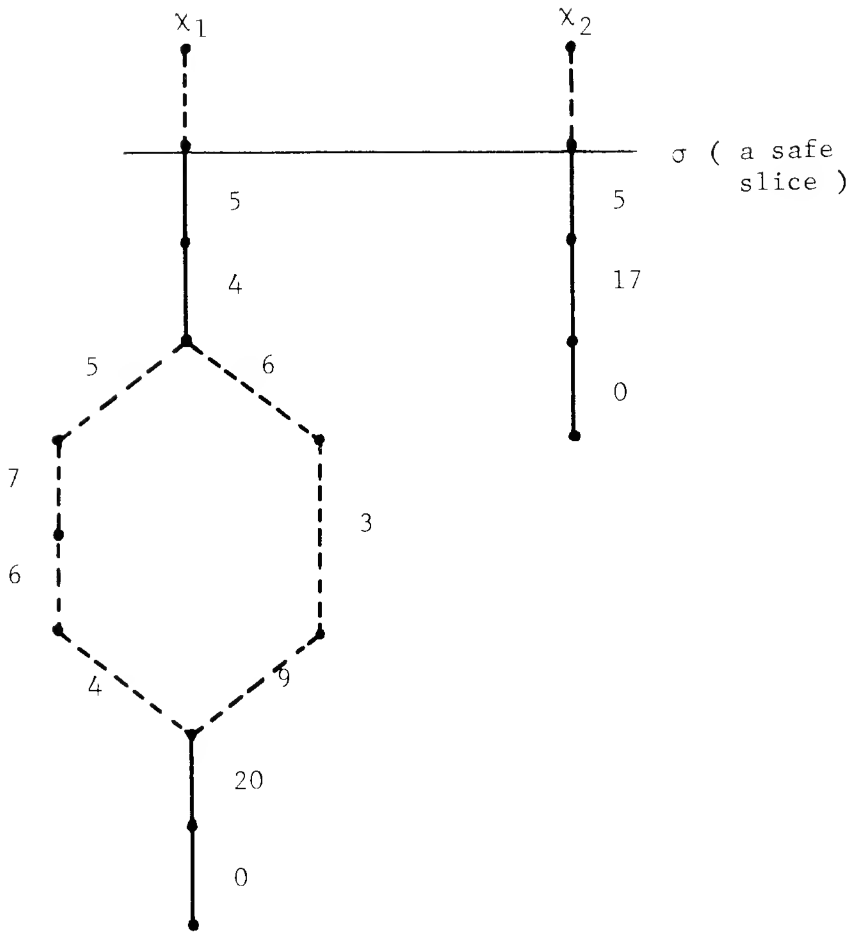
The choice of a representative chain depends on what is represented. If the point of choice represents a stage where a process autonomously chooses one path, then the representative chain should represent the "worst" alternative. If, on the other hand, the point of choice represents a stage in a process where one of several combinations of resources can meet its needs, so that the alternative paths represent the availability of choice to the resource allocator, then the "best" alternative is the one that should be represented. Since deadlock avoidance is of interest, the terms "best" and "worst" presumably represent the choices that are respectively most and least likely to make slices safe.

Unfortunately, which alternative is "best", say, depends on the slice being tested and, consequently, a local algorithm has to try all the alternatives one by one. This is illustrated in Figures 5.5 to 5.7. In Figure 5.5, slice γ_1 is safe only if the left hand alternative is used, while γ_2 is safe only if the right hand alternative is used. Figure 5.6 shows that even with scalar demands, the choice of an



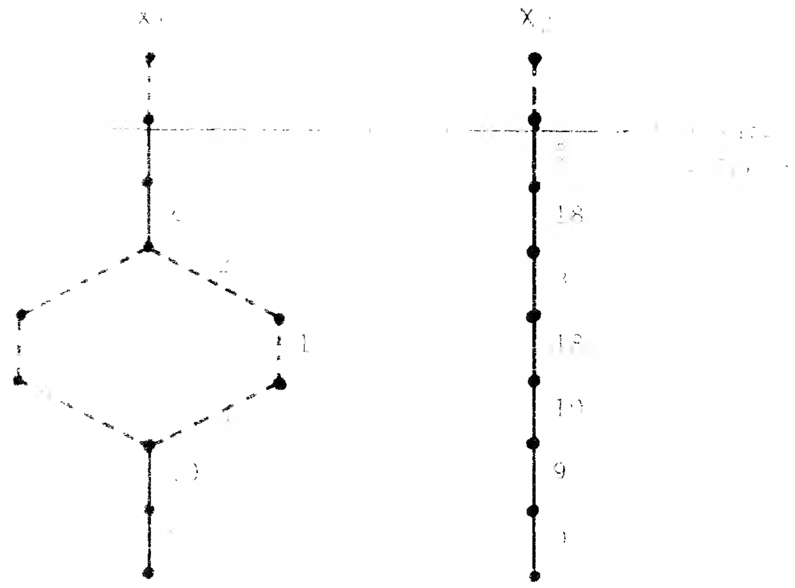
Capacity = (25,25)

Figure 5.5



Capacity = 20

Figure 5.6


X_1
X_2
2
4
1
10
8
18
12
10
9

[illegible]

[illegible]

alternative is not easy. In that figure the chain that has the larger maximum of demands on arcs is not inferior; for σ is safe only if the right hand alternative is used. Figure 5.7 shows that even if a chain has the smaller maximum of arc demands and the smaller minimum of arc demands, it can be "worse" than the other; for slice σ is safe only if the right hand alternative is used.

The selection of a "worst" alternative runs into similar problems.

Thus it is necessary to redefine a slice so that it is a set of arcs, one from each chain, with alternative chains emanating from a transition that represents a point of choice considered to be a single chain. Safeness algorithms have then to try the alternative chains one at a time until either a chain that can be crossed is found or all the chains can be crossed — the choice depends on whether the alternatives represent a decision by the process represented or a choice by the resource allocator. This, of course, increases the amount of backtracking and probably makes it non-linear.

Conclusion

Chapter 6

§6.1 Demand Graph Analysis of Resource Sharing — in Perspective

Deadlocks due to resource sharing are a result of limited resources and hoarding of allocated resources. In general, the avoidance of deadlock requires control of the acquisition of such resources by users, the entities that acquire and release resources. Total sequencing of the users, so that they proceed one at a time until completion, is always possible if no user ever needs more resources than are in the pool. Such control is gross and wasteful. Finer control requires information about resource usage by users.

The demand graph model is a model for the representation of information about resource usage by users when their activity can be divided into phases of known and steady resource usage. What scale of activity a phase represents can vary with the circumstances. The ability to represent a set of phases as a single phase whose demand is the least upper bound of the demands of the original phases is the key to this facility. The assumption of Habermann [3], that only the maximum demands of a user are known, corresponds to combining all the phases (other than the initial and terminal arcs) of the subgraph that represents the activities of a user and representing them by a single phase, whose demand is the least upper bound of the demands of all slivers of the sub-graph. It thus represents one extreme. However, there is a whole range of scales of representation on one side of that extreme, and demand graph analysis serves to illustrate what can be done in that range.

§6.2 Non-linearity in Algorithms

Between Scalar and Vector Demand Graphs there is a quantum jump in the amount of computation that a safeness algorithm has to do in the worst case. While it is to be expected that the amount of computation in the worst case increases as the number of components of demand increases, the increase would seem to depend more on the particular figures of demand encountered than on the number of components. For the Augmented Safeness Algorithm becomes non-linear only when it finds barriers before it finds arcs with total reduction in demand that satisfy the test of the Basic Algorithm; thus it is clear that the occurrence or non-occurrence of such lows of demand is what determines the amount of computation. However, the likelihood of occurrence of such lows in all components of demand may decrease as the number of types of resources in the systems represented increases.

It should be borne in mind that the non-linearity of the Augmented Safeness Algorithm is also a consequence of its local nature. The proof of non-linearity of the Augmented Safeness Algorithm assumed that even when barriers are discovered on all chains, the considerations for a slice to possess the prefix property must still be based on arbitrary extensions — not just those that also have barriers on all chains. This assumption was based on the particular definition used for local algorithms.

The principal cause for the non-linearity of the Augmented Safeness Algorithm is the fact that there are situations in which exactly one combination of crutches is useful and this can only be discovered by trial and error by a local algorithm.

The principal factor in the proof of non-linearity in arboraceous demand graphs even with scalar demands is the existence of situations in which a pre-synchronisation slice is accessible by exactly one sequence of chains on which to make moves. Here, too, if the demands on arcs incident on and emanating from points of synchronisation are small enough, then the amount of computation a safeness algorithm has to use does not become very large.

§6.3 Demand Graph Analysis in Operations Research

The problem of deadlocks is as serious in transportation, manufacturing, maintenance, etc., as it is in computer systems. That it has not been recognized in operations research is unfortunate, since the fields to which operations research addresses itself are those that are commonly encountered.

The assumptions for the demand graph model, viz that processes go through phases of known and steady resource usage, are particularly apt for manufacturing and other similar spheres to activity. The example of a maintenance hangar for aeroplanes in Chapter 1 is a case in point.

It should be pointed out that the assumption of an asynchronous nature for the processes in the analysis is not crucial, and its violation does not invalidate the results as far as scheduling problems in operations research are concerned. For deadlocks are caused by hoarding and improper coordination of the acquisition of resources by activities, not by the unpredictability of the durations of various phases of activity. That these durations are not known in asynchronous systems, merely implies that the activities should be viewed as discrete phases with a sequencing structure, rather than as continuous on-going activity.

The effect of knowledge of processing times or duration of phases is to make the various connected sequences of feasible slices from a safe slice to the terminal slice unequal — some sequences may be preferred over others, say because they result in a lower average running time for the processes. However, if a slice is not safe, then no schedule will allow all the processes to complete without deadlock. Thus considerations of deadlock prevention have the effect of eliminating certain schedules from the set of schedules that are considered for minimization of running time. All the work that has been done so far on selection of schedules that optimize running times can be applied to this reduced set.

§6.4 Use of Demand Graphs in Computer Systems

Deadlocks can occur in computer systems because processes commonly hoard resources such as locked data bases, main memory in systems with a single level memory, etc. Thus it would seem that demand graph analysis would be useful, and the next few sections touch upon some of the relevant issues.

The discussion in Chapter 1 pointed out that the scale for description of a computation (the activity of a "user" or a set of "users") as a sequence of phases can be chosen to suit the circumstances. Thus, it is possible to consider a phase as representing the execution of a single procedure or of a set of procedures, for instance. In other instances the phases may represent execution of parts of a procedure. The scale can, therefore, be chosen to suit the circumstances.

Although the discussion thus far has not touched on the effect of priorities, the use of priority schemes is not precluded. The analysis of Chapters 2 to 5 is invariant with choice of a priority scheme. It is perfectly reasonable to have any scheme, whatsoever, to select one or a few of several competing processes to receive resources, as long as allocating those resources corresponds to a move to a safe slice in the demand graph representation. In fact, one can even represent facilities such as guaranteed service, by modifying the safeness algorithms. If a certain sequential process needs to be guaranteed of always being able to proceed with the next k phases (for some value of k) as soon as it finishes the current one, then the safeness algorithm can be modified to assure such a process; for it can be made to ensure that every sequence of moves it constructs, during a test for safeness of any slice, begins with a uni-chain macro-move across k transitions on the chain representing that process. A page swapping process may perhaps be an instance of such a process.

Interactive systems are somewhat different. In such systems a user needs to be guaranteed not just of being able to complete his computation, but of being able to complete it within a reasonable amount of time. This "reasonable wait" constraint is usually quite strong and may imply either that the ability to preempt resources is required, no matter what the cost, or that computations should not be accepted until the expected time to completion is less than a certain limit. In such instances the analysis of demand graphs is still quite useful although, at times, only to provide guidelines or a philosophy, rather than to be applied directly and in detail.

System designers should not be distressed by the non-linearity of safeness algorithms for vector demand graphs. The large amounts of computation that non-linearity implies relate to worst cases and not necessarily to ordinary cases. Secondly, compromises are possible, since it is only required that the states that are permitted to occur are represented by safe slices not that all states that are represented by safe slices be permitted to occur. The cases in which the amount of computation begins to become rather large could be handled by refusal to consider the states represented by these slices for allocation.

Thus requests from processes for additional allocation may be denied because the slice representing the state that would result is unsafe, or because determination of its safeness takes too much computation. In this context the discussion in Section 6.1 on the scale of representation is quite relevant. In any case, the results in this thesis point out the sources of complexity and the degree of complexity that can be encountered. Non-optimal strategies[†] may be more practical and better, as long as extremes are avoided. The non-linearity of the Augmented Safeness Algorithm could thus be only of academic interest. Moreover, good heuristics could probably be found for commonly occurring situations.

Finally, there is a trend towards making resources preemptable on the one hand and effectively infinite on the other. The implementation of virtual memory schemes on multi-level memories is indicative of this trend. The trend is immensely desirable. However, deadlocks owing to sharing of locked data bases will continue to arise in computer systems, making coordinated allocation of such resources to avoid deadlock imperative.

[†]It should be pointed out that the Basic Algorithm in Chapter 3 only uses a sufficient condition (rather than a necessary and sufficient condition) as a test, anyway.

§6.5 Conclusions and Future Work

The demand graph model for the analysis of deadlocks is not the last word on the subject. Chapter 5 showed that the techniques of analysis become unmanageably cumbersome for unrestricted and augmented demand graphs. This is largely a consequence of the complex structures that these graphs exhibit. However, good algorithms for testing the safeness of slices of such graphs need to be devised and may require considerable ingenuity.

Moreover, there are several situations that demand graphs are incapable of representing. An output process for a group of recurring or cyclic processes that treats pieces of data from all processes symmetrically and operates with a finite buffer memory, which it shares with other output processes, is such an example. The full power of (unsafe) Petri nets [8] is required for the representation of such a system. For Petri net "conflicts" are required to represent the symmetrical treatment of pieces of data from all processes and the initiation of output as soon as possible after any such piece has arrived, without pre-ordained sequencing of the handling of outputs from the various processes served. This would suggest that Petri nets with numbers (demands) on places and constraints (capacity constraints) may be worth examining ab initio with a view to representing systems for analysis of deadlock.

In concluding, it should be pointed out that the work in this thesis represents an attempt to construct models for activities in systems so as to aid understanding and analysis of systems. Computer systems, in particular, need such models to aid in the understanding of fundamental problems. Such models are also required to provide tools for debugging of systems that are so complex that comprehension of the whole is almost impossible. The fact, that in using demand graphs to analyze consistency of use of locks on data bases one can construct the demand graph one process or one computation at a time, is of great value. For then mechanical tools (such as safeness algorithms) can handle the interactions of the parts in the complex whole. It is to be earnestly hoped that more debugging tools of this nature will be devised.

[illegible] the Theory of Linear Inequalities

To

Demand Graphs

Appendix

An examination of the lattice of slices of a demand graph such as that in Figure 2.4 shows an apparent redundancy of information. For instance, in the lattice of Figure 2.4 , all the arc labels have appeared in slice labels by rank 4. This suggests that a test of a kind different from that considered in the main body of this thesis may be possible. Such a test would utilize this observation, viz that the first few ranks of the lattice of slices contain a good deal of information. The test is, in general, a (K, p) feasibility test, i.e. a test which seeks p connected sequences of feasible slices from the test slice, γ, to a slice K ranks above γ in the lattice.

The test that is of particular interest is a (K, 1) feasibility test, especially because it is comparable to the tests discussed earlier. It should be interesting to determine how large K has to be in relation to L_γ, the rank of γ_T relative to γ. In determining such a lower bound on K, however, it is proposed to take a more mathematical approach in this appendix than has been taken so far. The intent of the analysis is to explore the effectiveness of such an approach rather than to obtain a tight bound for K. The investigation will therefore concern general questions such as what patterns of feasibility and infeasibility over the lattice of slices can be obtained, and so on. The mathematical tool that will be used is the theory of linear inequalities.

The reason why it appears, intuitively, that some patterns of feasibility and infeasibility may not be attainable is because these two types of constraints are opposite in nature and, therefore, could give rise to a contradiction. For example, suppose slices $A_2B_2C_2$ and $A_3B_3C_3$ of a three-chain demand graph were required to be feasible. Then could $A_2B_3C_3$ and $A_3B_2C_2$ both be infeasible? Clearly not, for the feasibility requirements imply that

$$d(A_2) + d(B_2) + d(C_2) \leq C$$

$$d(A_3) + d(B_3) + d(C_3) \leq C$$

$$d(A_2) + d(B_2) + d(C_2) + d(A_3) + d(B_3) + d(C_3) \leq 2C$$

$$\text{ie} \quad [d(A_2) + d(B_3) + d(C_3)] + [d(A_3) + d(B_2) + d(C_2)] \leq 2C$$

which clearly contradicts the infeasibility constraints.

In simple cases such as the example shown above, the incompatibility of the (four) constraints may be quite obvious. When a large number of constraints is involved, however, the incompatibility of constraints may be much less obvious. For this reason, it is proposed to seek simpler tests based on the exhibition of a well defined structure by the constraints. The principal task, then, is to determine what structures have important implications in this regard. It will be assumed that the demand graphs are rectilinear and that in any example the number of chains and the number of arcs on each chain are known, as is the capacity associated with the graph, but that values of demand that satisfy a given set of constraints are sought.

The discussion to follow assumes scalar demands but it is conjectured that the results are valid also for vector demand graphs.

Each requirement of feasibility of a slice imposes a constraint of the form

$$\sum_{i=1}^{m} a^j_{r_i} \leq C \quad \text{for the } j^{th} \text{ such slice}$$

where $a^j_{r_i}$ is the concise notation for $d(\alpha^i_{r_i})$ and the superscript j merely serves to identify the slice to which the inequality relates. Similarly, each requirement of infeasibility imposes a constraint of the form

$$\sum_{i=1}^{m} a^j_{r_i} > C \quad \text{or} \quad \sum_{i=1}^{m} (-a^j_{r_i}) < -C$$

The question of the compatibility of the feasibility and infeasibility requirements thus reduces to that of the consistency of a set of inequalities made up of inequalities of these two types. The theorem which follows relates to this question directly. It is taken from Cernikov [18].

"Theorem [3.4]. Let

$$f_j(x) - a_j \leq 0 \qquad j = 1, 2, 3, \ldots m$$

be an arbitrary compatible system of inequalities over the linear space[†] L(P) where P is an arbitrary ordered field[†], then the system

[†]See [9].

$$f_j(x) - a_j < 0 \qquad j = 1, 2, \dots m'; \quad m' \le m$$

$$f_j(x) - a_j \le 0 \qquad j = m' + 1, \dots m$$

$$x \in L(P)$$

is compatible iff the equation

$\Sigma u_j f_j(x) = 0$ with the unknowns $u_1, u_2, \dots u_m$

has no positive solutions satisfying the condition

$$a_1 u_1 + \dots + a_m u_m = 0 \; ; \quad u_1 + u_2 + \dots u_{m'} > 0\text{"}$$

In the discussion which follows, the linear space $L(P)$ is the linear space over the field of rational numbers since the components of demand are rational numbers.

An intuitive understanding of the theorem can be obtained by re-writing the inequalities as

$$f_j(x) < a_j \qquad j = 1, 2, \dots m'; \quad m' \le m$$

$$f_j(x) \le a_j \qquad j = m' + 1, m' + 2, \dots m$$

Each $f_j(x)$ is of the form $\beta_1 x_1 + \beta_2 x_2 + \dots \beta_n x_n$, where n is the dimension of the linear space $L(P)$. Since multiplying an inequality by a positive constant leaves the inequality unaltered, if positive mul-tipliers u_j can be found which (after multiplication) make the sum of the left hand sides identically zero, then in a compatible system the corresponding sum of the right hand sides must be greater than zero (unless no non-zero multiplier multiplies any inequality in the second group), or else one gets the absurd conclusion $0 < 0$! What is less obvious, and therefore interesting, is that this condition is also sufficient for compatibility.

Now, a given pattern of feasibility and infeasibility implies that a set of inequalities be true simultaneously. This set is

$$A_0 \begin{cases} \sum_{i=1}^{m} a^j_{r_i} \leq C & j \in [1, p'] \quad \text{for the } p' \text{ feasible slices} \\ \sum_{i=1}^{m} (-a^j_{r_i}) < -C & j \in [p' + 1, p] \quad \text{for the } p-p' \text{ infeasible slices} \end{cases}$$

The theorem quoted above is applicable to this set of inequalities only if it is compatible when the inequality in the second group is changed to "$\leq$". But this is clearly true, since a value of $\frac{C}{m}$ for each $a^j_{r_i}$ results in satisfaction of all of the resulting inequalities. Therefore, the theorem is applicable to the set, A_0, of inequalities given above.

In order to apply the theorem to the inequalities in A_0, a correspondence of terms must be set up. Consider one of the inequalities in A_0:

$a_{r_1} + a_{r_2} + \dots + a_{r_m} \le C$ (The j merely identifies the slice from which the inequality comes and hence which a_{r_i}'s appear)

The variables here are the demands a_{r_i}. Let the total number of distinct demand variables appearing in A_0 be N. Then the above inequality is of the form

$$[1, 0, 0, \dots 1, 0, \dots] \begin{bmatrix} \\ a \\ \\ \end{bmatrix} \le C$$

where the column vector a is the vector of N demand variables, and the row vector, with m components having a value 1, serves to pick out those components of a which appear in the inequality above. Thus a corresponds to the x of the theorem. The row vector of 1's and 0's is called a <u>selection vector</u>.

Now the equation

$$\sum_{j=1}^{m} u_j f_j(a) = 0$$

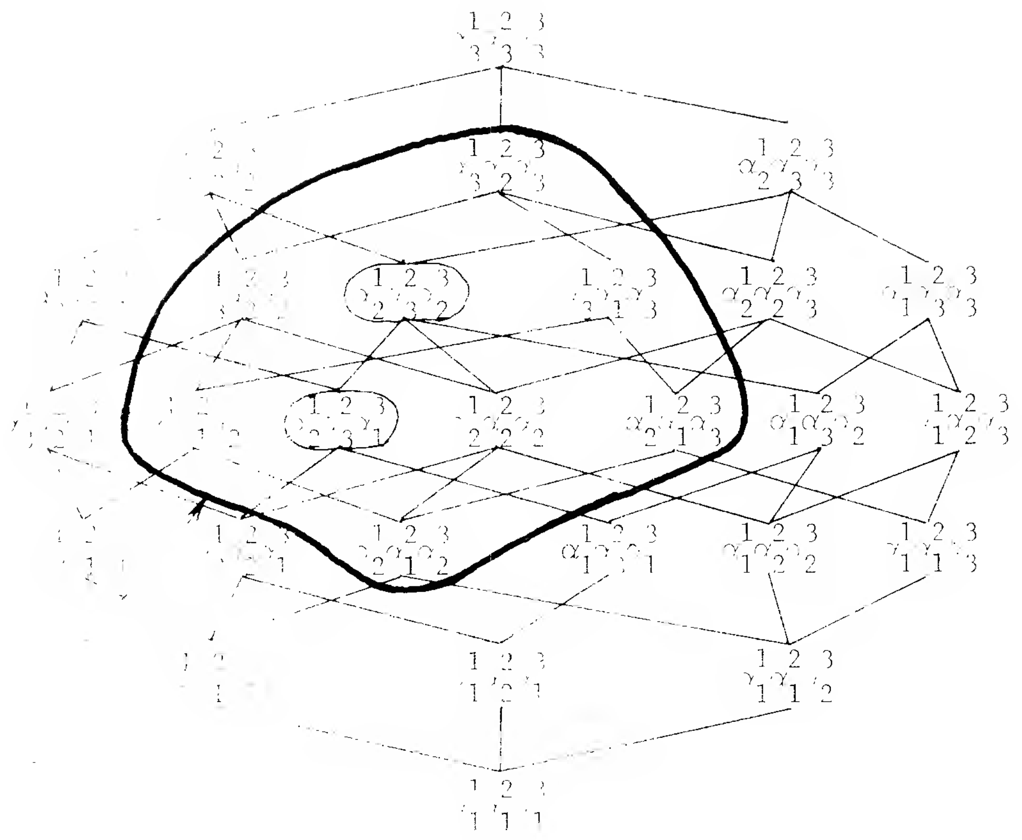
in the variables u_j is really an identity in terms of a, since the equation has to be true for all values of a.

The two lemmas which follow interpret the implications of the theorem stated above in terms of two patterns of feasibility and infeasibility. The patterns are described in terms of a substructure of the lattice of slices called a hull, a precise definition of which appears later.

A sub-lattice is a subset of the elements of the lattice which is itself a lattice under the same definitions for computing least upper bounds and greatest lower bounds. It can be shown that a sub-set of a lattice that is closed under the operations of the lattice is a sub-lattice. Consequently, one can generate a sub-lattice from any sub-set of a finite lattice by adding the elements needed to make the set closed.

The <u>hull</u> of a set, A, of slices is the set of all slices, σ, in the lattice that satisfy g.l.b.(A) $\preccurlyeq \sigma \preccurlyeq$ l.u.b.(A). Figure 1 shows the hull of a set of slices. The hull of a set of slices is a sub-lattice of the lattice of all slices since the hull is closed with respect to the operations of extracting the greatest lower bounds and least upper bounds of slices. It is clear that every slice in the hull of a set of slices, A, lies on a directed path from g.l.b.(A) to l.u.b.(A).

<u>LEMMA 1</u> Let D be a demand graph of m chains with n_i arcs on the i^{th} chain. Let $\gamma_1, \gamma_2, \dots \gamma_p$ be slices of D required to be infeasible and $\gamma_{p+1}, \dots \gamma_q$ be slices required to be feasible. Then a set of demands for the arcs of the demand graph that ensure that all these conditions are met exists if none of the slices $\gamma_{p+1}, \dots \gamma_q$ lies



Hull of [illegible] and $\gamma_2^1\gamma_1^2\gamma^3$ (Encircled elements are not in the hull)

Figure 1

in the hull of $\gamma_1, \ldots \gamma_p$ (or symmetrically, if none of the slices $\gamma_1, \ldots \gamma_p$ lies in the hull of $\gamma_{p+1}, \ldots \gamma_q$).

PROOF: The system of inequalities whose consistency is being examined consists of the two sets of inequalities:

$$\sum_{i=1}^{m} -a^j_{r_i} < -C \qquad j \in [1, p] \text{ -------------------- } (1)$$

corresponding to the infeasible slices, $\gamma_1, \gamma_2, \ldots \gamma_p$, and

$$\sum_{i=1}^{m} a^j_{r_i} \leq C \qquad j \in [p+1, q] \text{------------------ } (2)$$

corresponding to the feasible slices, $\gamma_{p+1}, \gamma_{p+2}, \ldots \gamma_q$.

Step 1: Suppose that positive multipliers $\lambda_1, \lambda_2, \ldots \lambda_p$ and $\mu_{p+1}, \ldots \mu_q$ for the two sets exist such that

$$\lambda_1 \left(\sum_{i=1}^{m} -a^1_{r_i}\right) + \lambda_2 \left(\quad \right) + \ldots \lambda_p \left(\quad \right)$$

$$+ \mu_{p+1} \left(\sum_{i=1}^{m} a^{p+1}_{r_i}\right) \ldots\ldots\ldots\ldots + \mu_q \left(\quad \right) \equiv 0$$

i.e.,

$$\lambda_1 \left(\sum_{i=1}^{m} a^1_{r_i}\right) + \ldots + \lambda_p(\ ) \equiv \mu_{p+1} \left(\sum_{1}^{m} a^{p+1}_{r_i}\right) + \ldots + \mu_q(\ ) \text{--- } (3)$$

Then, for consistency, the theorem requires that

$$(-C) \cdot \sum_{j=1}^{p} \lambda_j + (C) \cdot \sum_{k=p+1}^{q} \mu_k > 0$$

be true unless λ_j is 0 for all values of j.

Consider the identity in (3). It will be noticed that as each term in parentheses relates to a slice, it contains exactly m variables, each with coefficient 1. Since one can multiple (3) through by the LCM of the denominators of the λ's and μ's to get integer multipliers, it may be assumed that the λ's and μ's are integers so that one can speak of the number of terms on one side of (3). The number of terms appearing on the left hand side when (3) is expanded out is $m \sum_{1}^{p} \lambda_j$ whereas that on the right hand side is $m \sum_{p+1}^{q} \mu_k$. Since (3) is an identity, it is necessary, inter alia, that these two numbers be equal. Thus

$$\cancel{m} \sum_{1}^{p} \lambda_j = \cancel{m} \sum_{p+1}^{q} \mu_k$$

Therefore,

$$-C \sum \lambda_j + C \sum \mu_k = 0$$

The system of inequalities consisting of (1) and (2), therefore, is inconsistent if positive integer values for the λ's and μ's exist that satisfy (3).

The identity in (3) can be rewritten in terms of the selection vectors so as to eliminate the variables. It then becomes the set of N equations

$$[-\lambda_1 \quad -\lambda_2 \quad \cdots \quad -\lambda_p] \begin{bmatrix} \text{selection vector for } \gamma_1 \\ \cdot \\ \cdot \\ \cdot \\ \text{selection vector for } \gamma_p \end{bmatrix}$$

$$= [\mu_{p+1} \quad \mu_{p+2} \quad \cdots \quad \mu_q] \begin{bmatrix} \text{selection vector for } \gamma_{p+1} \\ \cdot \\ \cdot \\ \cdot \\ \text{selection vector for } \gamma_q \end{bmatrix}$$

Now both sides of this identity can be multiplied by the N X 1 vector of α's which corresponds to a. One then gets an identity which is identical to (3) but with arc-labels rather than demands in it. Call the new identity (4) — it is an identity of algebraic expressions whose terms are the arc labels $\alpha^i_{r_i}$.

That values for the λ's and μ's that satisfy (3) should not exist, implies (in terms of the identity (4)) that values for the λ's and μ's that satisfy (4) should not exist. That is, for consistency no permutation of the collection of labels of slices in a selection (with repetition) from the set $\{\gamma_1, \gamma_2, \ldots \gamma_p\}$ should produce a collection of labels that is also a permutation of the collection of labels of slices in a selection of the same size from $\{\gamma_{p+1}, \dots \gamma_q\}$ (again with repetition allowed); selection of a slice more than once corresponds to a multiplier γ (or μ) which is greater than one.

<u>Step 2</u>: Now consider a selection Σ (in this proof, always with repetition allowed) from $\{\gamma_1, \dots \gamma_p\}$. Then any permutation of the collection of labels of these slices that yields slice-labels must satisfy the condition that each component $a^i_{r_i}$ of a new slice label satisfies:

g.l.b. of the arc numbers r_i of the χ_i components of slice labels in Σ $\le$ r_i $\le$ l.u.b. of the arc numbers r_i of the χ_i components of slice labels in Σ

therefore,

g.l.b. of the χ_i arc components of the slices in the selection Σ $\le$ $\alpha^i_{r_i}$ $\le$ l.u.b. of the χ_i arc components of the slices in the selection Σ

since the arcs on each chain are numbered in sequence downwards. Thus

g.l.b. of the slices from the selection $\le$ the slice in question (resulting from a permutation) $\le$ l.u.b. of the slices from the selection

i.e., the slice lies in the hull of $\{\gamma_1, \dots \gamma_p\}$.

(For example one slice-label resulting from the permutation of $\alpha^1_3\alpha^2_2\alpha^3_2$ and $\alpha^1_2\alpha^2_3\alpha^3_2$ is $\alpha^1_3\alpha^2_3\alpha^3_2$, which clearly satisfies $\alpha^1_1\alpha^2_2\alpha^3_2 \le \alpha^1_3\alpha^2_3\alpha^3_2 \le \alpha^1_3\alpha^2_3\alpha^3_4$, and $\alpha^1_3\alpha^2_3\alpha^3_2$ does lie in the hull of $\alpha^1_3\alpha^2_2\alpha^3_2$, $\alpha^1_2\alpha^2_3\alpha^3_2$ and $\alpha^1_1\alpha^2_2\alpha^3_4$.)

Thus, if none of the slices $\gamma_{p+1}, \ldots \gamma_q$ lies in the hull of $\gamma_1, \ldots \gamma_p$, then the collection of labels in any selection from $\{\gamma_{p+1}, \ldots \gamma_q\}$ cannot be a permutation of the collection of labels in a selection from $\{\gamma_1, \ldots \gamma_p\}$. Thus no inconsistency can result and, therefore, demands for the arcs exist so that both the feasibility and infeasibility requirements are satisfied.

Q.E.D.

LEMMA 2. Consider a demand graph with m chains, the ith chain having n_i arcs. Let $\{\gamma_1, \ldots \gamma_p\}$ be a set of slices of the demand graph which lie in one rank, R, and let $\gamma_1, \ldots \gamma_p$ be required to be infeasible. Furthermore, let $\gamma_1, \ldots \gamma_p$ completely partition their hull, i.e. there does not exist a slice at rank R that is in the hull of $\{\gamma_1, \gamma_2, \ldots \gamma_p\}$ but is not in $\{\gamma_1, \ldots \gamma_p\}$. Then if values of demand can be found so as to make the slices $\gamma_1, \gamma_2, \ldots \gamma_p$ infeasible and all the slices below rank R in the hull of $\{\gamma_1, \gamma_2, \ldots \gamma_p\}$ feasible, then no slice that lies above rank R in this hull can be feasible.

PROOF: Let γ be a slice in the hull at a rank greater than R that is required to be feasible. Let $\{\gamma_{j_1}, \gamma_{j_2}, \ldots \gamma_{j_\ell}\}$ be a minimal subset of $\{\gamma_1, \ldots \gamma_p\}$ such that γ is the l.u.b. of $\gamma_{j_1}, \ldots \gamma_{j_\ell}$. That is, $\{\gamma_{j_1}, \ldots \gamma_{j_\ell}\}$ is the smallest set of slices at rank R whose l.u.b. is γ. Such a set has to exist since the set of all slices at rank R that lie on a directed path from g.l.b. $\{\gamma_1, \ldots \gamma_p\}$ to γ certainly have γ as a l.u.b., and all such slices belong to the hull.

Now it will be shown that slices $\gamma_{s_1}, \ldots \gamma_{s_{\ell-1}}$, all lying above rank R, exist such that the labels of $\gamma, \gamma_{s_1}, \gamma_{s_2}, \ldots \gamma_{s_{\ell-1}}$ are permutations of the labels of $\gamma_{j_1}, \gamma_{j_2}, \ldots \gamma_{j_\ell}$.

In the discussion that follows the labels of slices $\gamma, \gamma' \ldots$ will be designated by $\gamma, \gamma' \ldots$. This should cause no confusion as the context should resolve any ambiguity. The labels $\gamma_{s_1}, \ldots \gamma_{s_\ell}$ are obtained as follows: Take out the elements that make up γ from $\gamma_{j_1}, \gamma_{j_2}, \ldots \gamma_{j_\ell}$. Then take any of the ℓ "stripped" labels remaining and distribute its components among the other ℓ-1 stripped labels, giving each a component that is from the same chain as the one it contributed to γ. The resulting labels are $\gamma_{s_1}, \gamma_{s_2}, \ldots \gamma_{s_{\ell-1}}$.

(The construction is illustrated for $A_3\ B_1\ C_2$, $A_2\ B_3\ C_1$ $A_1\ B_2\ C_3$ below:

$\gamma_{j_1}, \gamma_{j_2}, \gamma_{j_3}$:	$A_3\ B_1\ C_2$	$A_2\ B_3\ C_1$	$A_1B_2C_3$
γ:		$A_3\ B_3\ C_3$	
Stripped labels:	$B_1\ C_2$	$A_2\ C_1$	$A_1\ B_2$
Result of distribution:		$A_2\ B_1\ C_1$	$A_1\ B_2\ C_2$

Clearly, $(A_3 + B_1 + C_2) + (A_2 + B_3 + C_1) + (A_1 + B_2 + C_3)$

$= (A_3 + B_3 + C_3) + (A_2 + B_1 + C_1) + (A_1 + B_2 + C_2)$.)

In general, it is obvious that $\gamma, \gamma_{s_1}, \ldots \gamma_{s_{\ell-1}}$ is a permutation of $\gamma_{j_1}, \gamma_{j_2}, \ldots \gamma_{j_\ell}$.

It remains to be shown that $\gamma_{s_1}, \gamma_{s_2}, \ldots \gamma_{s_{\ell-1}}$ all lie above rank R and are therefore feasible — this, together with the feasibility of γ and the infeasibility of $\gamma_{j_1}, \gamma_{j_2}, \ldots \gamma_{j_\ell}$ leads to an inconsistency.

It is obvious from the construction that

Each component of a completed stripped label	$\leq$	The corresponding component of γ

For each label, γ_s, some one component the relation is really "<" — this is the component received in the distribution, i.e. the one component the unstripped label alone can contribute to γ.

(E.g., B_1 (in A_2 B_1 C_1) $\leq B_3$ (in A_3 B_3 C_3) above)

Thus each of the resulting slices $\gamma_{s_1}, \gamma_{s_2}, \ldots \gamma_{s_{\ell-1}}$ has an index sum less than R.

Q.E.D.

<u>LEMMA 3</u>. If a slice γ of a demand graph is feasible but none of its immediate successors is, then no slice other than γ in the hull of its successors can be feasible.

<u>THEOREM</u> Let D be an m-chain demand graph and let L be the rank in the lattice of slices of its terminal slice, γ_T. Let there exist a connected sequence of feasible slices from a slice γ of D to a slice, γ', at rank L-m and let γ' have m successors. Then the sequence can be extended to γ_T.

Lemma 3 follows from the fact that γ is the only slice in the hull of its successors that lies above the successors. The theorem follows from Corollary 1; for the hull of the m successors extends m-1 ranks below themselves, and thus encompasses γ_T — therefore at least one of the successors must be feasible (from Lemma 3) and one can apply the result to the successors of that slice, and so on.

The theorem above is a result , regarding a (K, 1) safeness test, of the kind that was sought at the beginning of the appendix. Undoubtedly, many more results of this kind could be proved. The aim of the appendix, however, is merely to indicate the nature of results that can be obtained by utilization of the theory of linear inequalities.

REFERENCES

1. Crowston, W. B. S., Decision Network Planning Models, Doctoral Dissertation, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, Pa., May 1968.

2. Habermann, A. N., "Prevention of System Deadlocks", Communications of the ACM, July 1969, pp 373-378.

3. Habermann, A. N., On the Harmonious Cooperation of Abstract Machines, Doctoral Dissertation, Department of Mathematics, Technological University, Eindhoven, The Netherlands, 1967.

4. Dijkstra, E. W., De Bankiersalgorithme, Department of Mathematics, Technological University, Eindhoven, The Netherlands, 1965.

5. Havender, J. W., "Avoding Deadlock in Multitasking Systems", IBM Systems Journal, No. 2, 1968, pp 74-85.

6. Shoshani, A. and Coffman, E. G., Sequencing Tasks in Multiprocess, Multiple Resource Systems to Avoid Deadlock, Technical Report Number 78, Princeton University, June 1969.

7. Shoshani, A. and Coffman, E. G., Detection, Prevention and Recovery from Deadlocks in Multiprocess, Multiple Resource Systems, Technical Report No. 80, Department of Electrical Engineering, Princeton University, October 1969.

8. Holt, A. W., Information System Theory Project, Final Report, RADC-TR-68-305, Rome Air Development Centre, New York, 1968.

9. Birkhoff, G. and MacLane, S., Algebra, The MacMillan Company, New York, 1967.

10. Dijkstra, E. W., <u>Cooperating Sequential Processes</u>, Department of Mathematics, Technological University, Eindhoven, The Netherlands, 1965.

11. Conway, R. W. et al, <u>Theory of Scheduling</u>, Addison Wesley Publishing Company, Reading, Massachusetts, 1967.

12. Harary, F. et al, <u>Structural Models</u>, John Wiley and Co., New York, 1965.

13. Berge, C., <u>The Theory of Graphs</u>, Translation, Methuen and Co. Ltd., Great Britain, 1962.

14. Conway, M. E., "A Multiprocessor System Design," Proc. Fall Joint Computer Conference, 1963, pp 139-146.

15. Corbato, F. J. and Vyssotsky, V. A., "Introduction and Overview of the Multics System," AFIPS Conference Proceedings, Volume 27, 1965, pp 185-196.

16. Van Horn, E. C., <u>Computer Design for Asynchronously Reproducible Multiprocessing</u>, Doctoral Dissertation, Department of Electrical Engineering, M.I.T., August 1966.

17. Hebalkar, P. G., <u>Asynchronous Cooperative Multiprocessing Within Multics</u>, S.M. Thesis, Department of Electrical Engineering, M.I.T., June 1968.

18. Ĉernikov, S. N., "Algebraic Theory of Linear Inequalities", American Mathematical Society Translations, Series 2, No. 69, 1968, pp 147-203.

INDEX

arbour 101
arc 21
  initial 22, 101
  terminal 22, 101
accessible 58
barrier 44, 71
Basic Algorithm 77
capacity 21
chain 22
chain-graph 105
crutch 71
Crutch Algorithm 78
current slice 34
demand 21
demand graph 21
  arboraceous 101
  manifold 150
  rectilinear scalar 21
  rectilinear vector 51
  scalar 21
  vector 51
directed path 30
earlier than or equal to 28
efflorescence 127
extension 57
feasible slice 33
Fundamental Algorithm 129
fork 101
Fork Algorithm 134
frustulum 105
frustum 105
hull 173
immediate successor 28
later than or equal to 28
lattice 30
  distributive 30
  of slices 31
Lemma 3.1 59
  3.2 63
  3.3 65
  3.4 89
  4.1 113
  4.2 115
  4.3 116
limited-backtracking algorithm 88
linear algorithm 88
local algorithm 87
macro-move 33
move 33
non-linear algorithm 88
partial ordering 29
partially ordered set 29
point of interaction 101
point of synchronisation 101
post-synchronisation slice 127
prefix property 58
prefix relation 58
pre-synchronisation slice 127
safe slice 33
Safeness Algorithm 43
  Augmented 75
  General 128
  Modified 57
segment of a chain 22
set of extensions 57
slice 24, 103
  initial 24
  terminal 24
sliver 103
Theorem 2.1 44
  3.1 67
  3.2 80
  3.3 92
  4.1 117
  4.2 120